Overview and Tour of the Mission Data System

♦

Bob Rasmussen, chief architect
Dan Dvorak, deputy architect

August 6, 2002

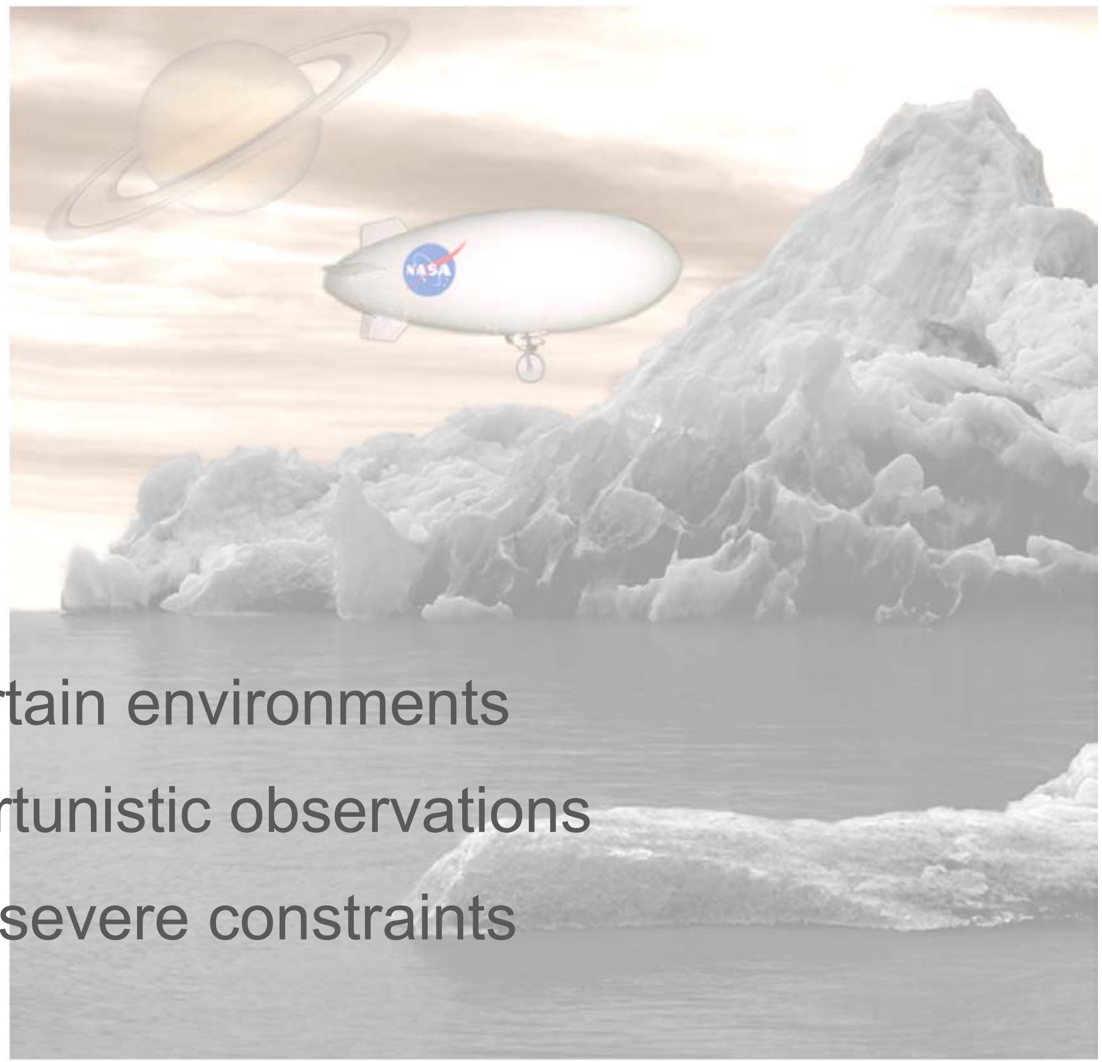
Outline

- Overview
  - Motivations, vision, themes
- A Virtual Tour
  - A balloon ride down into the depths of MDS
- State Analysis Process
  - Questions and answers about how things work
- MDS Implementation Status
- Questions & answers

Challenging Future Missions

- In situ exploration
- Multi-vehicle coordination
- More complex observatories
- An interplanetary network

Pressures for Autonomy



- Uncertain environments
- Opportunistic observations
- More severe constraints

More Complex Tasks

- Interact with and alter the environment
- Alter plans to achieve objectives
- Coordinate multiple competing activities
- Manage resources
- intelligently screen data before transmission

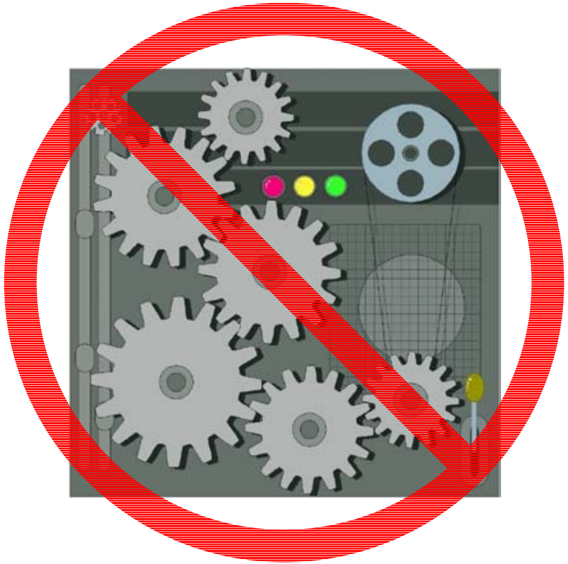
Meeting These Needs Will Be Difficult

Historical Gaps

- Between expressing **what operators want** and expressing **how to get it**
- Between **flight** and **ground** software developments
- Between missions in **inheritance** of flight software
- Between ground generated **time-based sequencing** and **fault tolerance**
- Between **systems** and **software** engineering

Pressures for Change

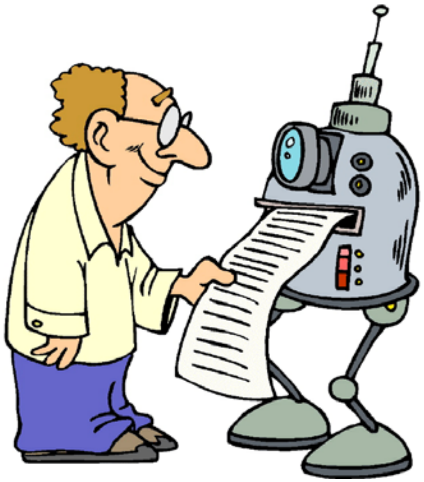
- New era of more frequent launches
- Demands for lower cost
- Specter of mission-ending failures due to errors in software
- Success must be assured, despite large uncertainties



No

What We Need

- Highly reusable core software for flight, ground, and test
- Synergistic systems & software engineering
- Reduced development time and cost
- Improved development processes
- Highly reliable operations
- Increased autonomy



Yes

The MDS Vision

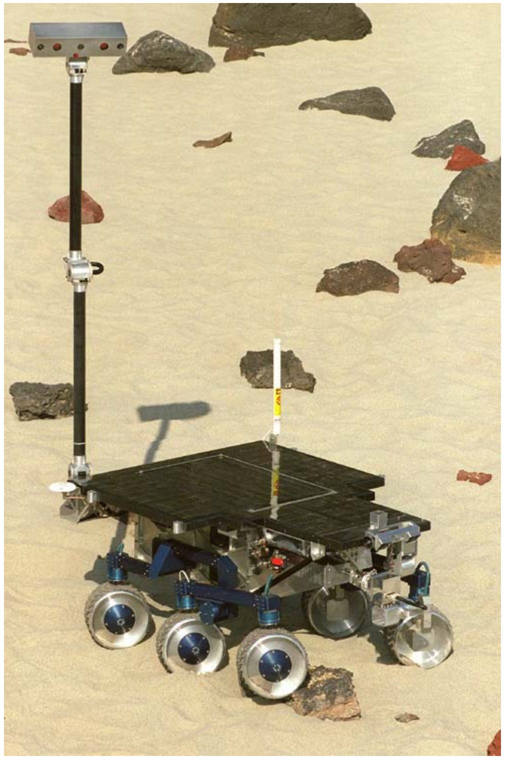
A unified control architecture and methods
for flight, ground, and test systems
that enable missions
requiring reliable, advanced software

What is MDS … Really?

- An **architecture**, unifying flight, ground, & test systems
- An orderly **systems engineering methodology**
- Software **frameworks** (C++ and Java)
- **Processes**, tools, and documentation
- Working **examples** of adaptation
- **Reusable software** components

Achieving the Vision

- MDS project started in 1998
- Initial focus on analysis and prototyping
- Full implementation initiated in 2001
- Design and implementation of core frameworks near completion — continuing updates
- Current focus on maturation through application to real systems
- Running on Rocky 7 and FIDO rovers
- Baselined for Mars Smart Lander project



MDS Themes

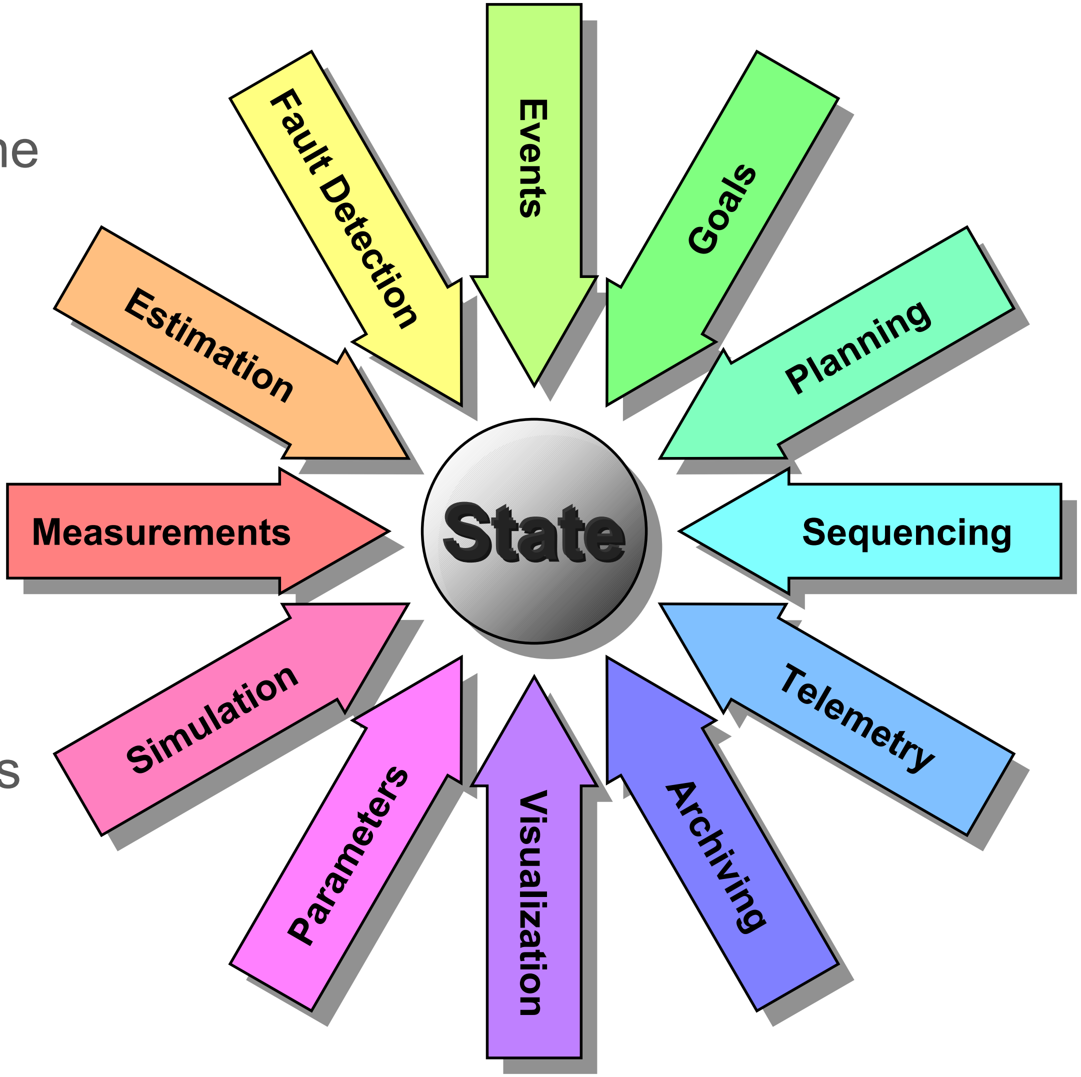
- Construct subsystems from architectural elements, not the other way around

A unified approach to managing interactions is essential

- Some things go together — others do not
- Be explicit (use goals, models, …)
- Close the loop
- Think ahead

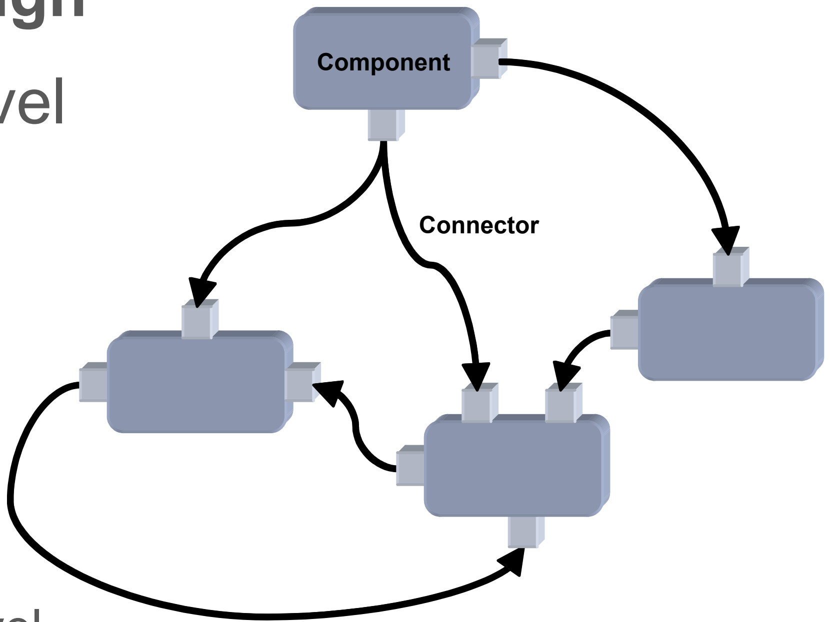
A State-Based Approach

- Formally recognize **state** as the key system concept
- Make **state** the central organizing theme for most functions
- Express all knowledge in models of **state**
- Share a common definition of system **state** among models



A Component-Based Approach

- The **State Architecture** establishes the elements of **functionality**, but *not* the software design
- The **Component Architecture** establishes the elements of **software design**
- Issues are raised to the level of symbolic realization
  - Software is organized as **components**
  - State-based elements are realized as **components**
  - Complexity of interactions is managed at the **component** level



Complementary Approaches

State-Based
Architecture

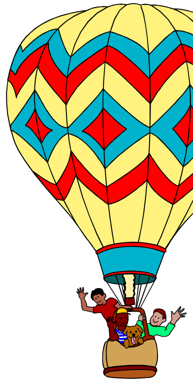
- Handles interactions among elements of the system under control
- Outward looking
- Addresses systems engineering issues

Component-Based
Architecture

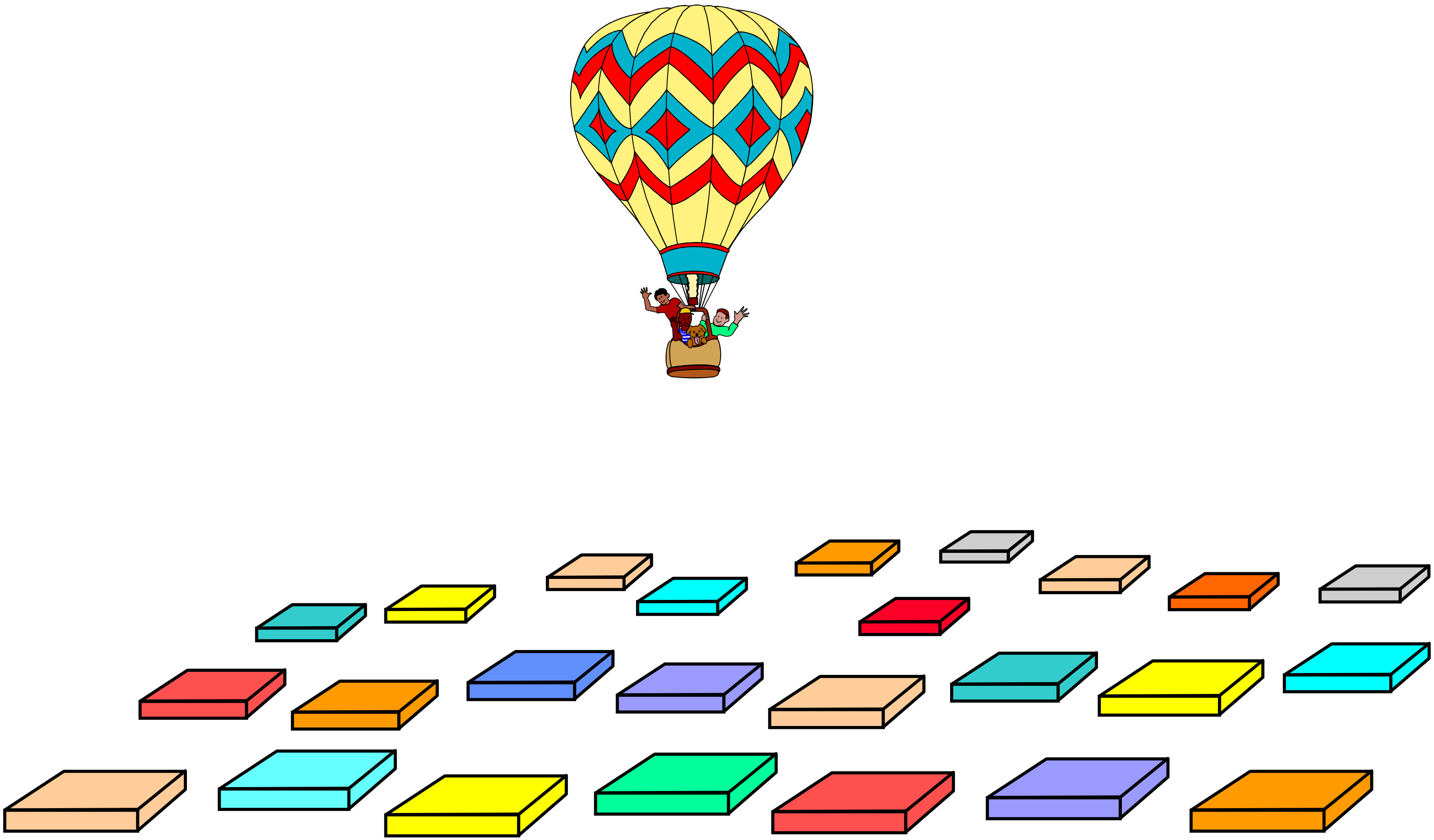
- Handles interactions among elements of the system software
- Inward looking
- Addresses software engineering issues

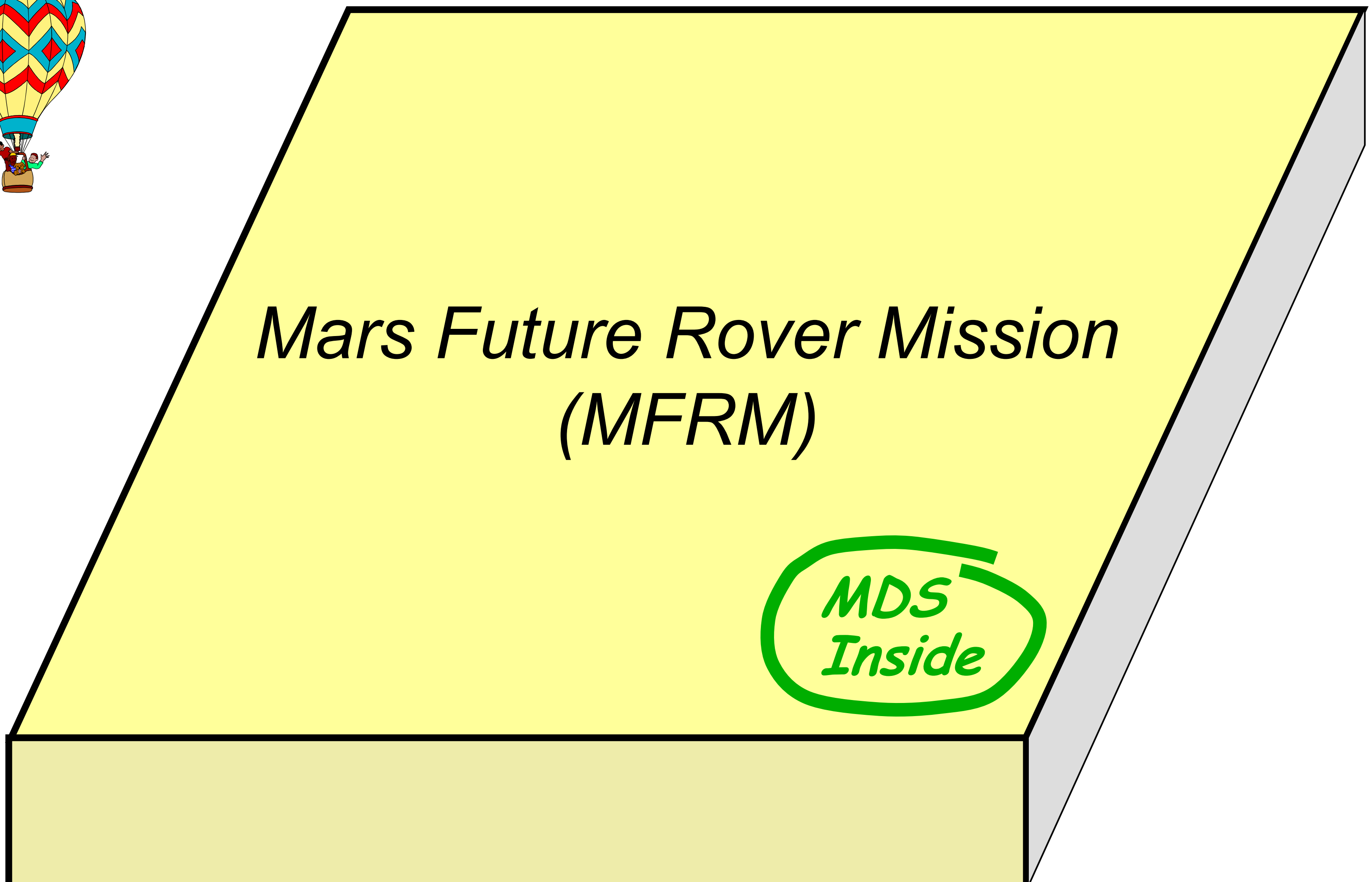
A Virtual Tour

Descending into a MDS-based system

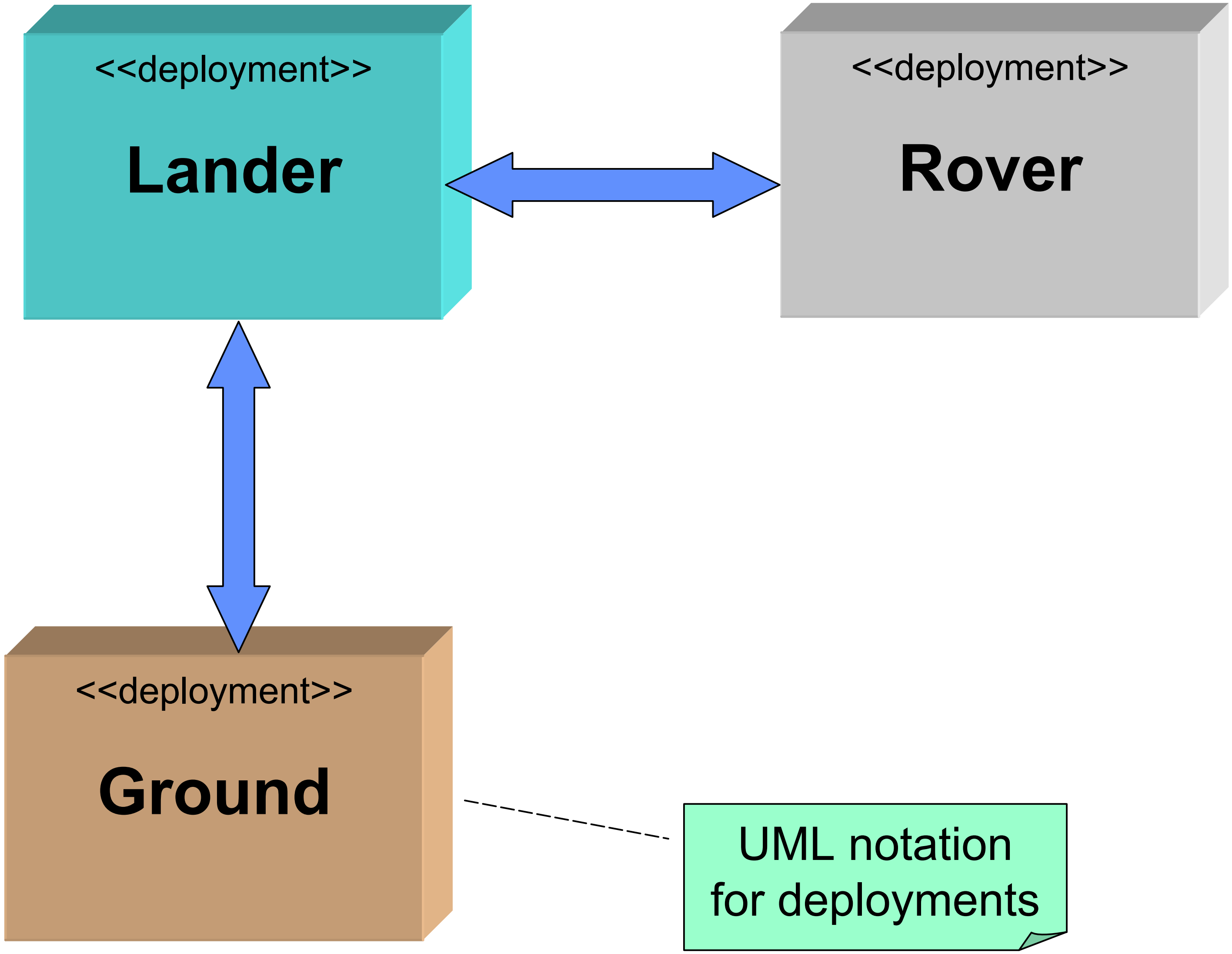


25 Missions in Next 10 Years




Mars Future Rover Mission
(MFRM)
MDS
Inside

Inside Mission MFRM



Inside Lander

<<discipline>>
Science

<<discipline>>
Data Products

<<discipline>>
Power

<<discipline>>
GNC

<<discipline>>
Thermal

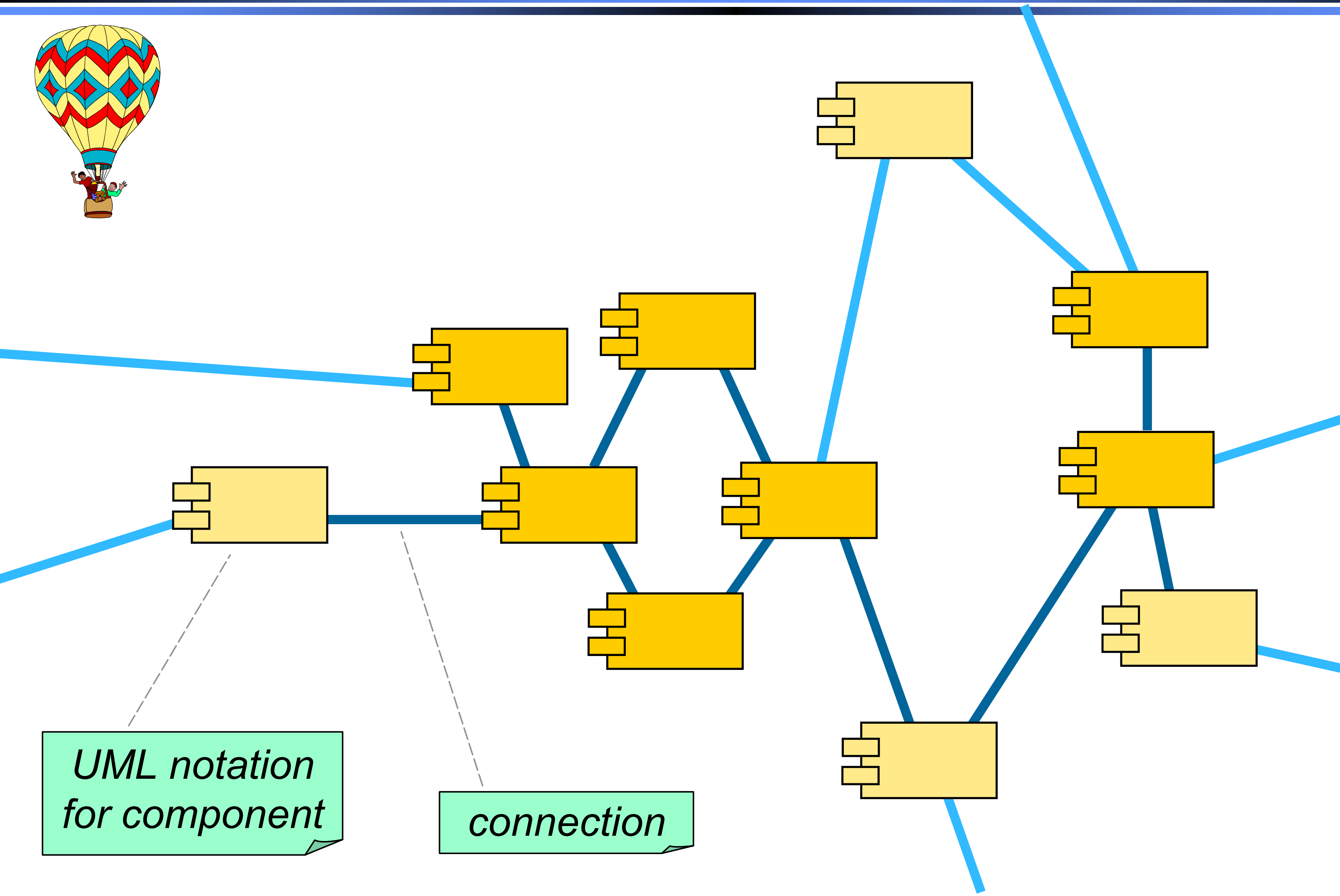
<<discipline>>
Avionics

<<discipline>>
Pyro

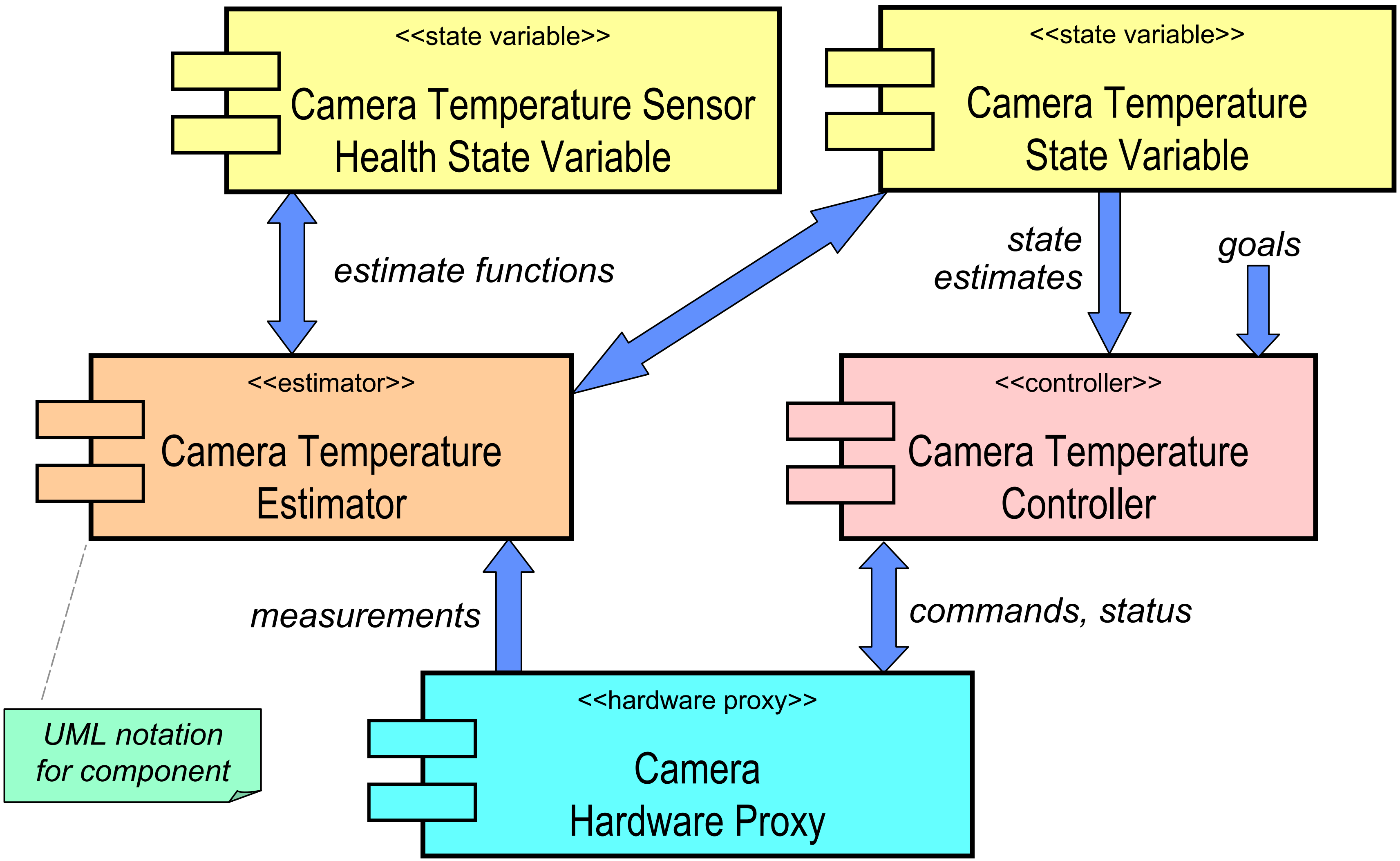
<<discipline>>
Telecom

Next: Thermal

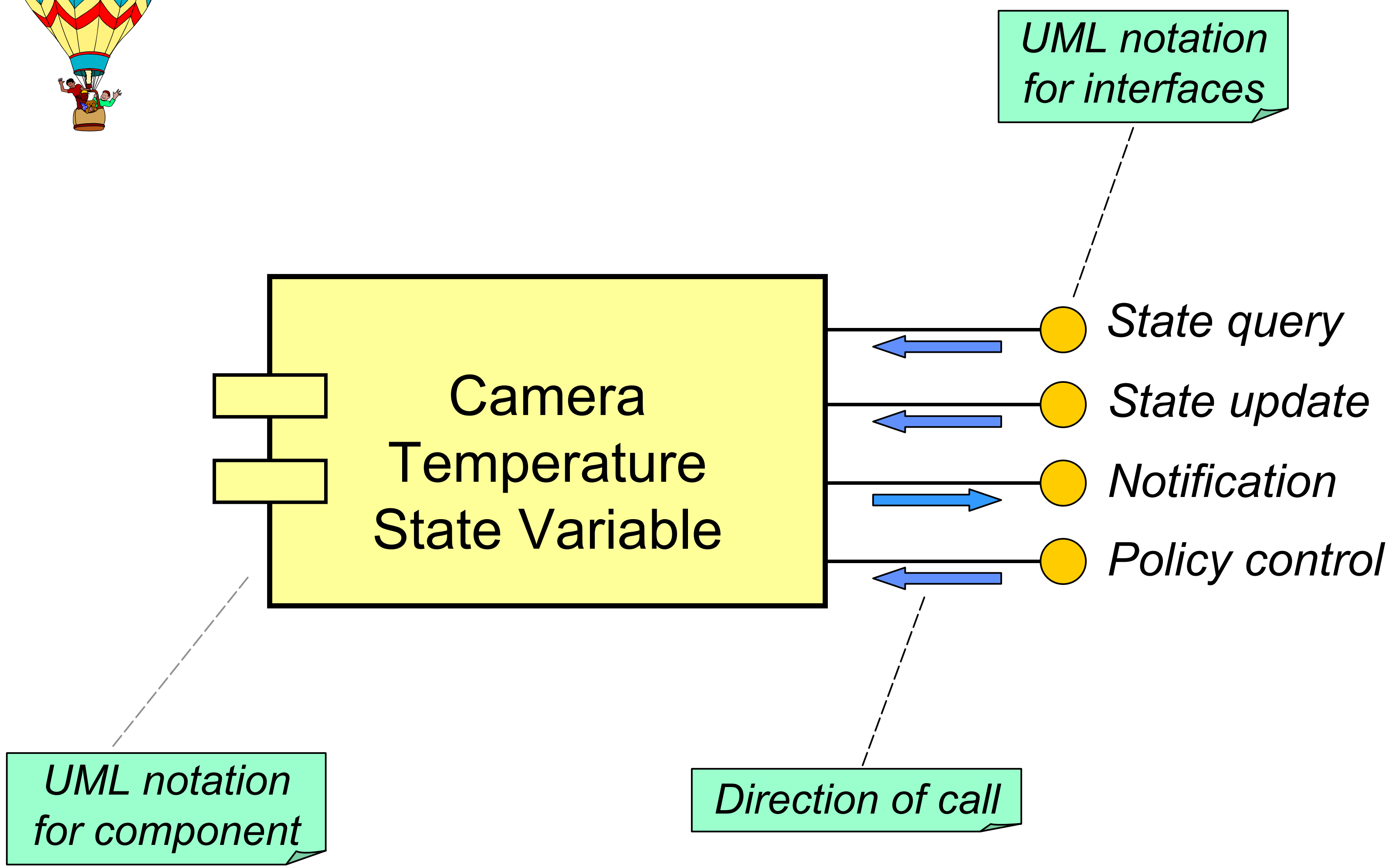
Inside Thermal, Depth 1



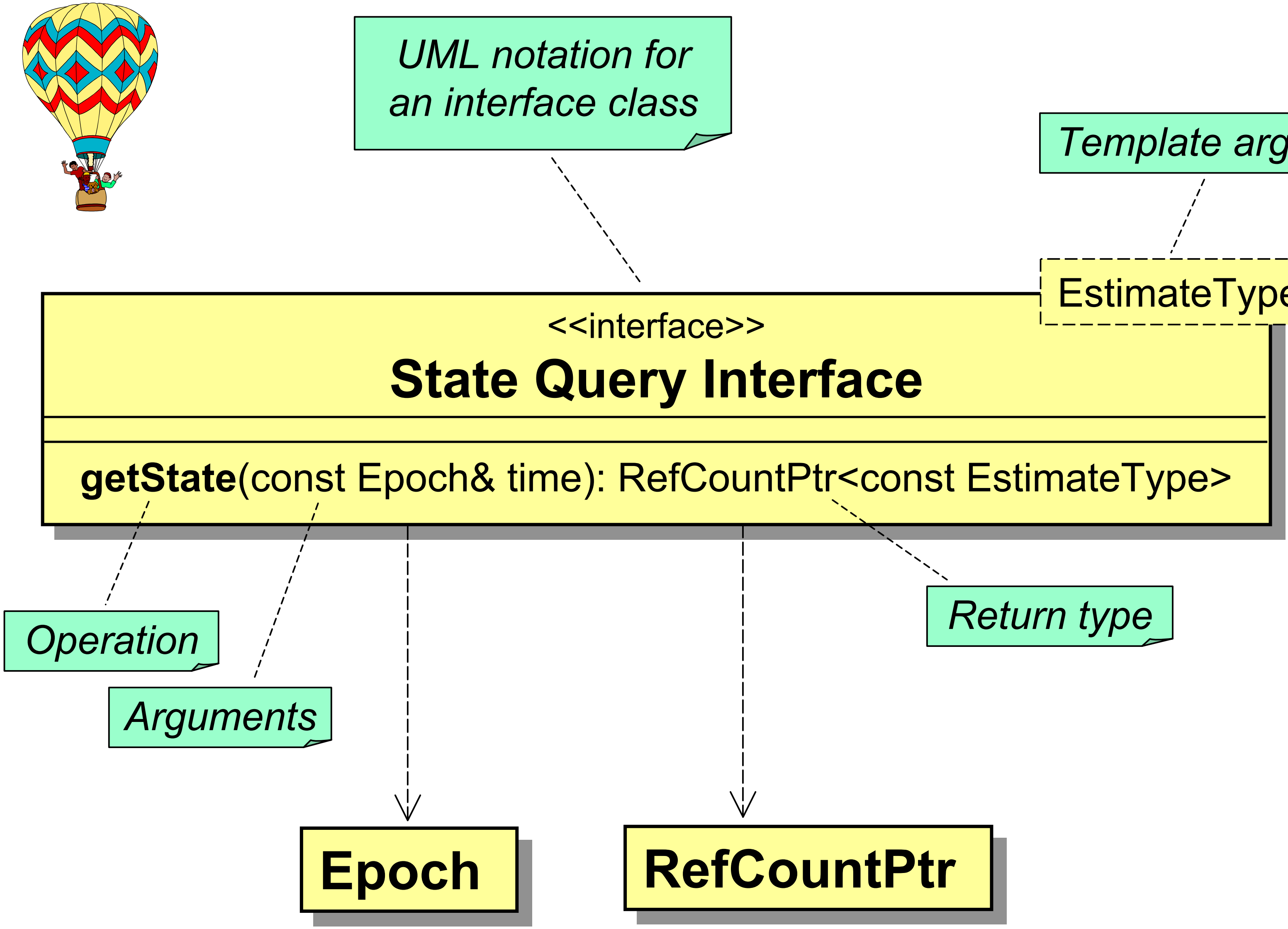
Inside Thermal, depth 2



Outside a State Variable



Looking at an Interface



JPL
MFRM
Lander
Thermal
- depth 1
- depth 2

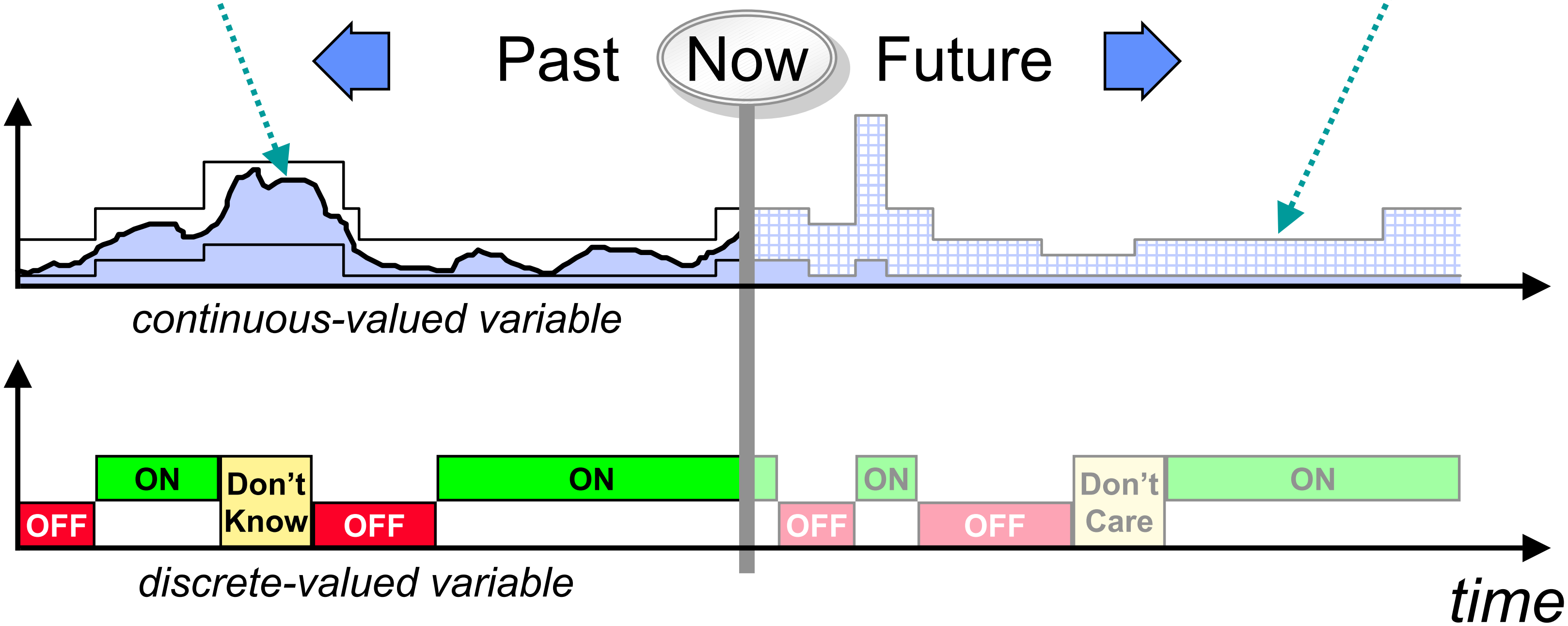
state var
-outside
-interface

Inside a State Variable

A timeline represents a state variable's value as a function of time

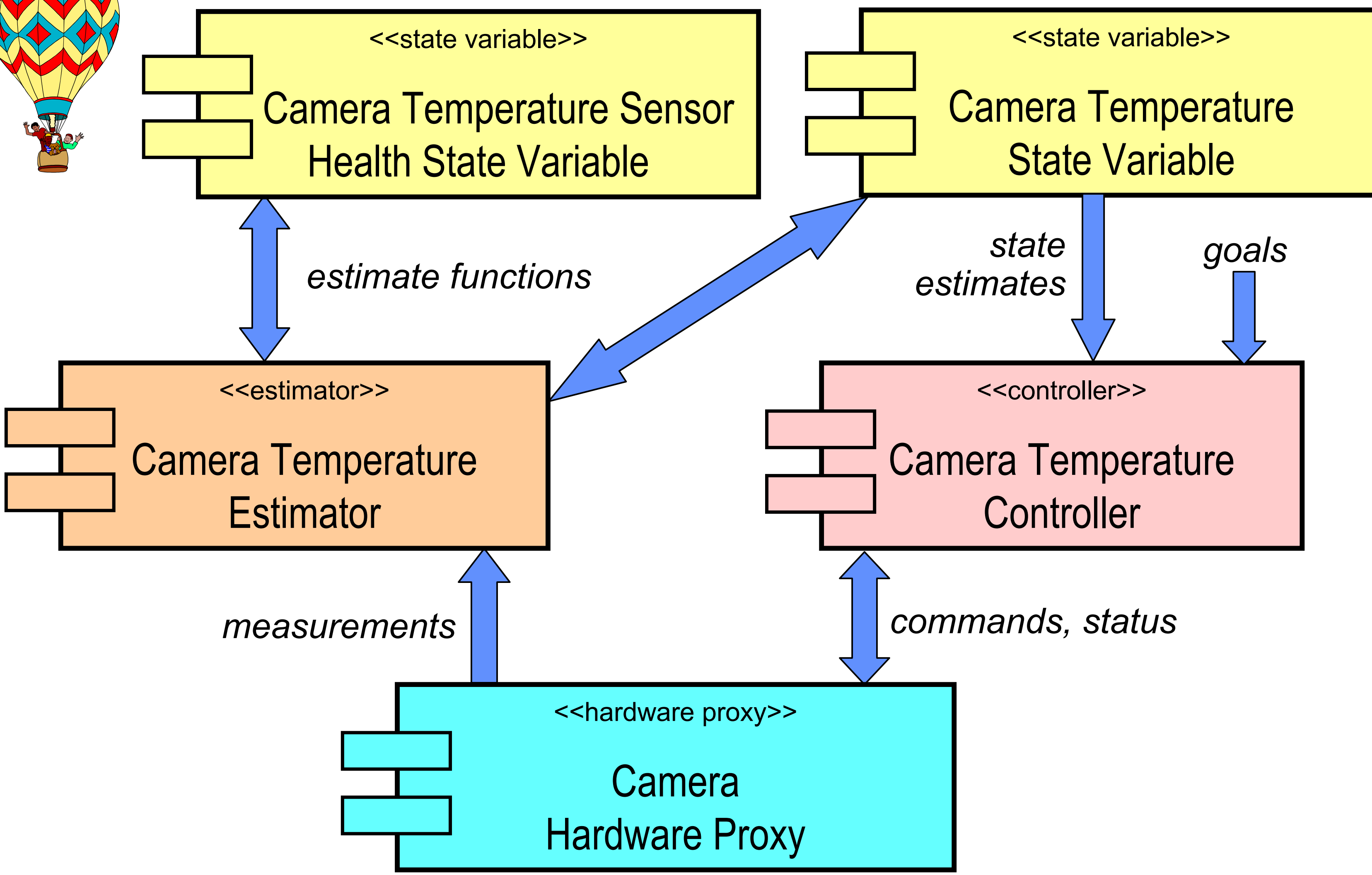
Estimates states (knowledge)

Planned states (intent)



Inside Thermal, depth 2

JPL
MFRM
Lander
Thermal
- *depth 1*
- *depth 2*



State Analysis

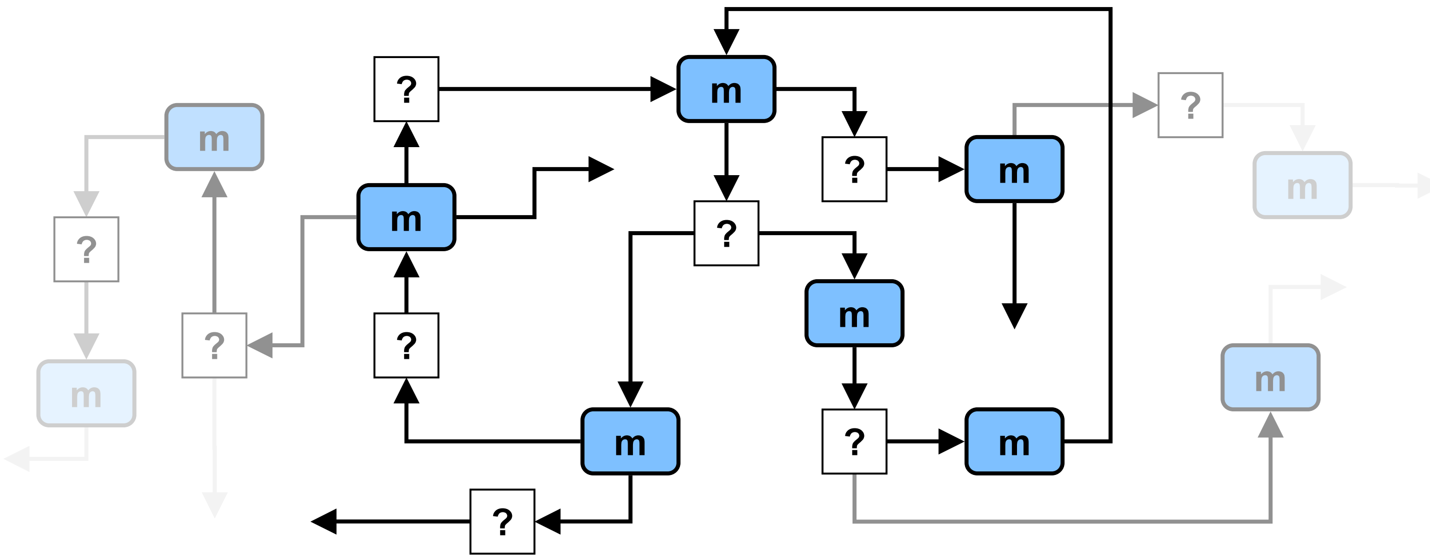
A gradual, methodical discovery process …

… that systems engineers use …

… to ask and answer questions about how things work

State Analysis is Recursive

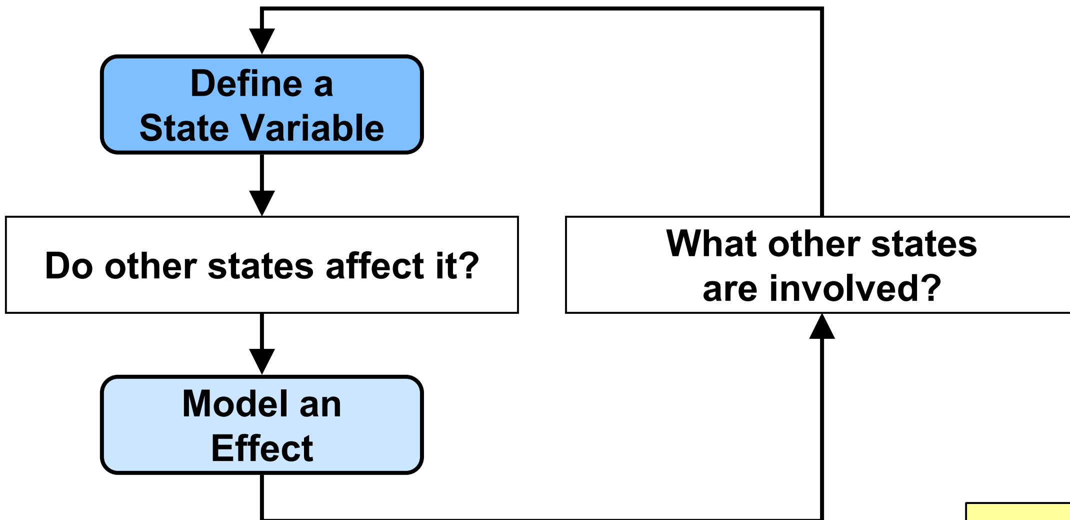
- It is a gradual discovery process, prompted by a **standard** set of basic **questions**
  - The answer to each question is a piece of the model



  - Each answer prompts additional questions, and so on
  - The model unfolds a step at a time in terms of **common framework elements** until all the pieces are identified

State Variables

Understanding the System in Terms of State



- Start with a few key states
  - Look at their behaviors
  - Ask how and why they change
- Revisit this for every new state variable that is identified

Each model element gets at least a name and a description.

Most have several other characteristics and links to other elements that must be described.

Spacecraft States

- Dynamics
  - Vehicle position & attitude, gimbal angles, wheel rotation, …
- Environment
  - Ephemeris, light level, atmospheric profiles, terrain, …
- Device status
  - Configuration, temperature, operating modes, failure modes, …
- Parameters
  - Mass properties, scale factors, biases, alignments, noise levels, …
- Resources
  - Power & energy, propellant, data storage, bandwidth, …
- Data product collections
  - Science data, measurement sets, …
- DM/DT Policies
  - Compression/deletion, transport priority, …
- Externally controlled factors
  - Space link schedule & configuration, …

… and so on

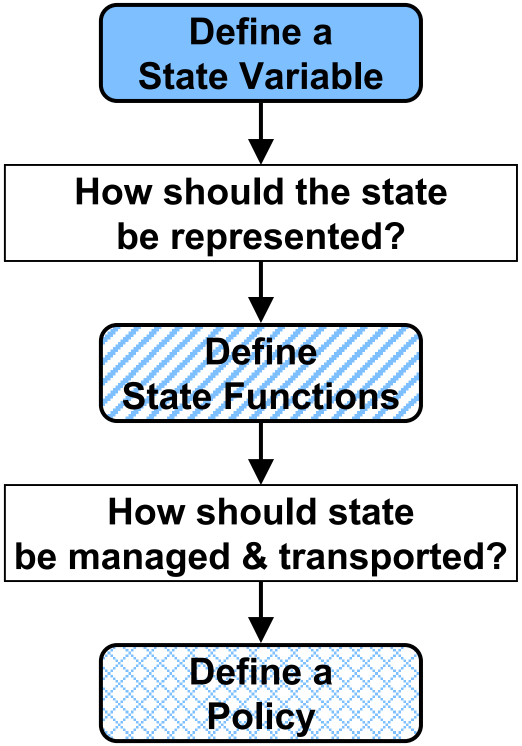
Spacecraft Models

- Relationships among states
  - *Power varies with solar incidence angle, temperature, & occultation*
- Relationships between measurement values and states
  - *Temperature data depends on temperature, but also on calibration parameters and transducer health*
- Relationships between command values and states
  - *It can take up to half a second from commanding a switch to full on*
- Sequential state machines
  - *Some sequences of valve operations are okay; others are not*
- Dynamical state models
  - *Accelerating to a turn rate takes time*
- Inference rules
  - *If there has been no communication from the ground in a week, assume something in the uplink has failed*
- Conditional behaviors
  - *Pointing performance can't be maintained until rates are low*
- Compatibility rules
  - Reaction wheel momentum cannot be dumped while being used for control

… and so on

State Value Histories

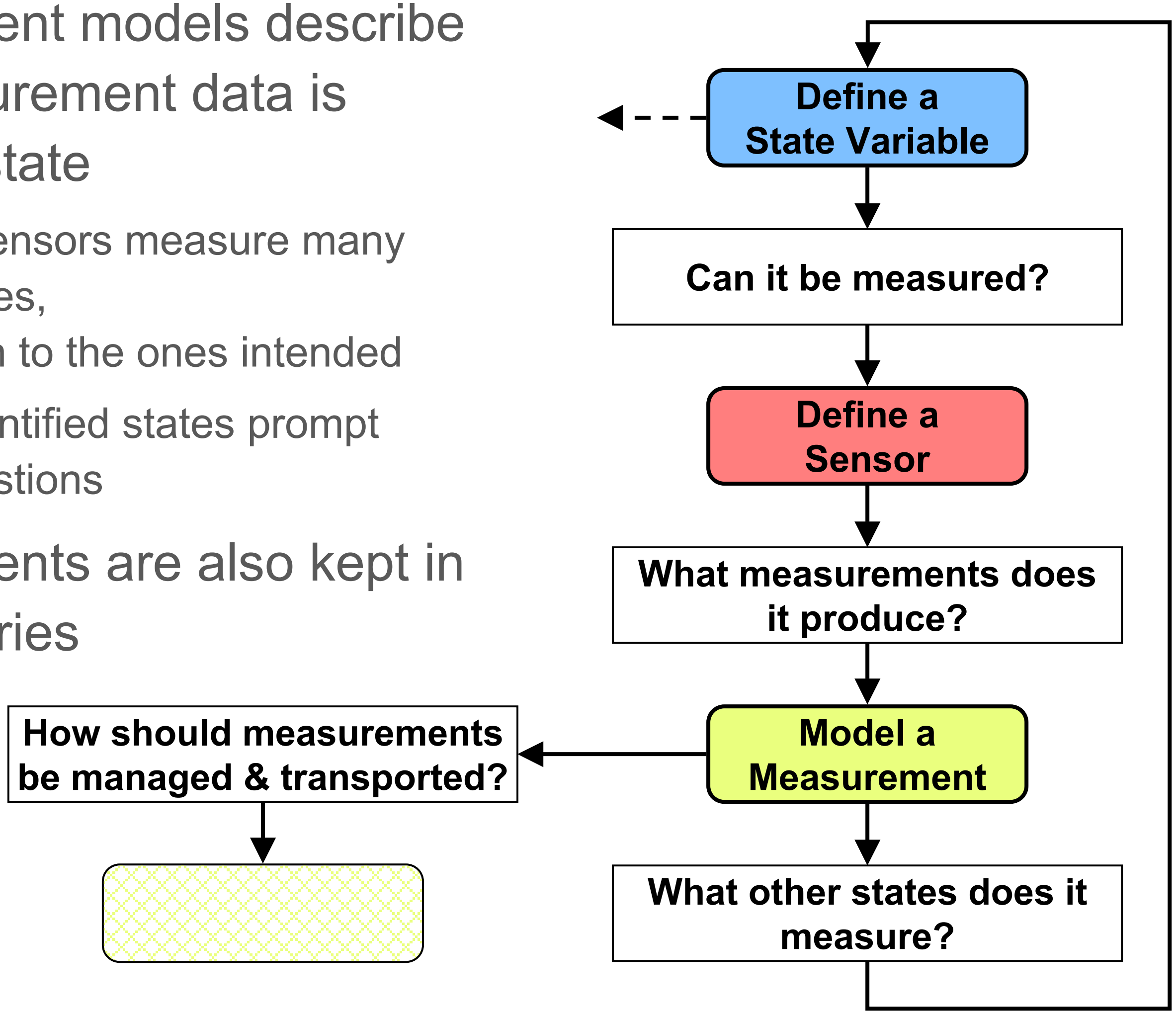
Reporting What's Happening in Terms of State



- Choose carefully how each state will be expressed
  - This is driven by need
  - Uncertainty must be part of the definition
- Value histories maintain state values as functions of time
  - They record the past
  - They may also predict the future
  - They are transported across space links to report what is happening
- Policies guide the treatment of this data
  - This includes converting it into new forms

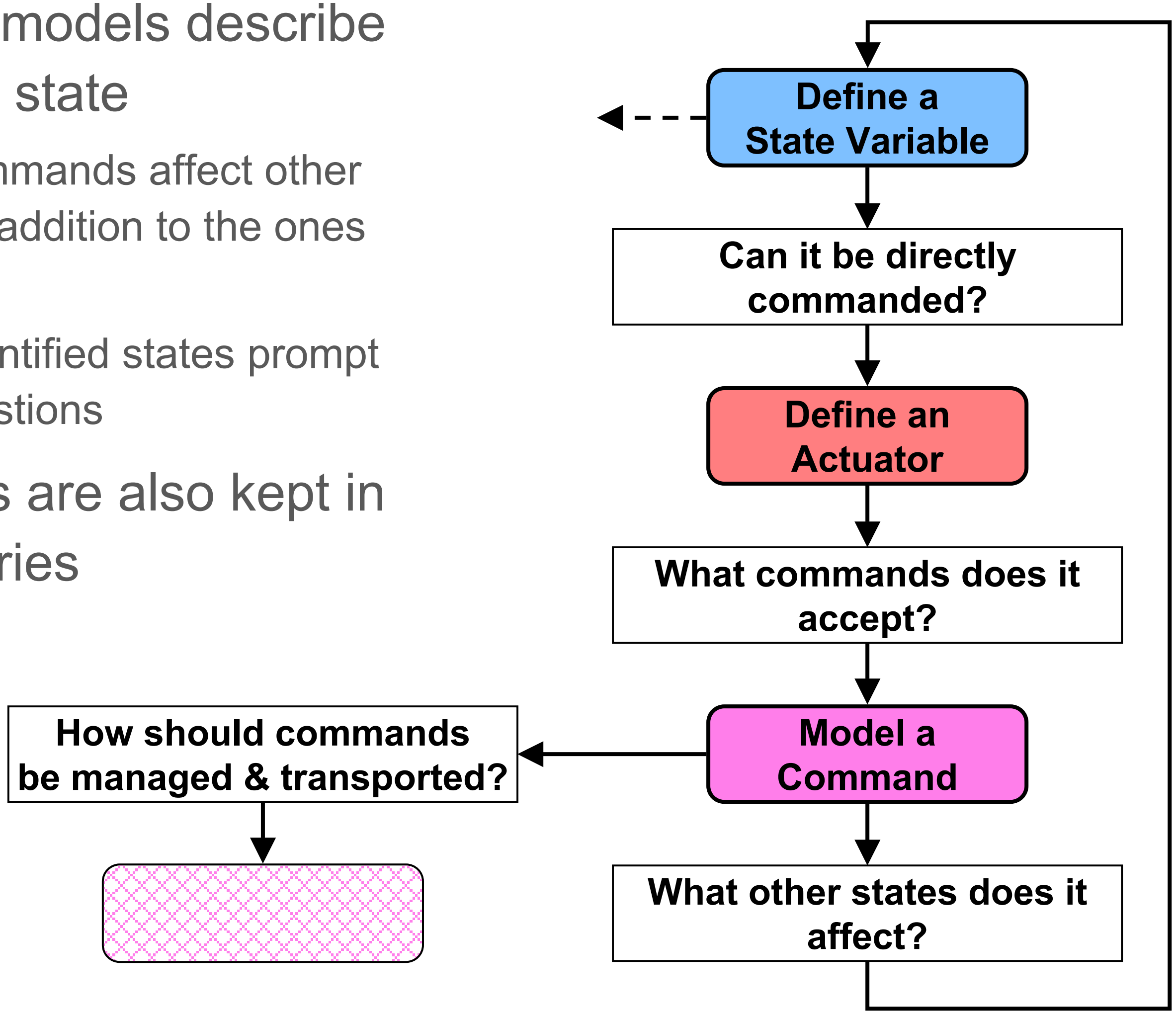
Sensors (Input Devices)

- Measurement models describe how measurement data is related to state
  - Usually sensors measure many more states,
    in addition to the ones intended
  - Newly identified states prompt more questions
- Measurements are also kept in value histories



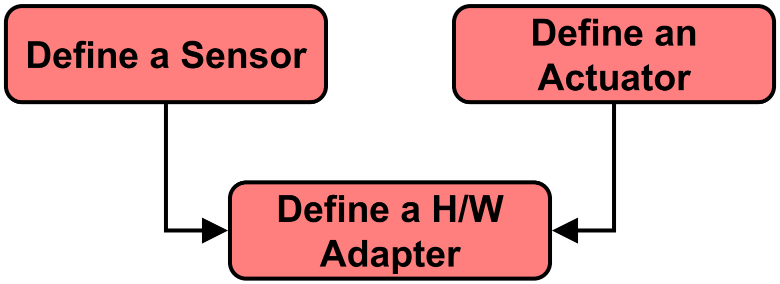
Actuators (Output Devices)

- Command models describe changes to state
  - Often commands affect other states, in addition to the ones intended
  - Newly identified states prompt more questions
- Commands are also kept in value histories



Hardware Adapters

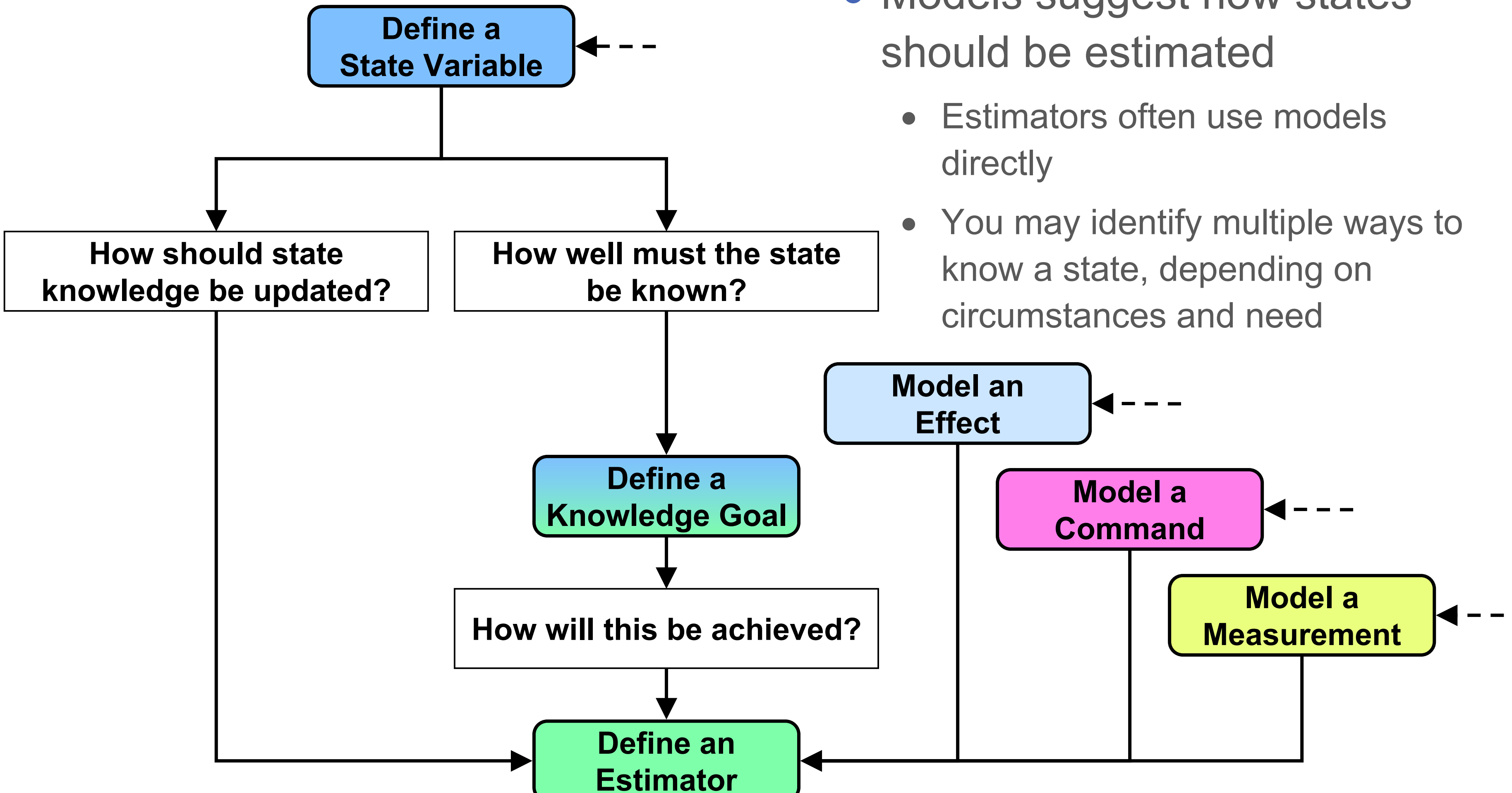
- Sensors and actuators are input and output ports, respectively on hardware adapters
  - Hardware adapters handle all communication with the hardware
  - They may also augment hardware capabilities with various low level services
- Collectively, these hardware and service functions present architecturally uniform sensor and actuator ports to the rest of the software



State Determination

Monitoring the System and Its Own Actions to Determine State

- Models suggest how states should be estimated
  - Estimators often use models directly
  - You may identify multiple ways to know a state, depending on circumstances and need

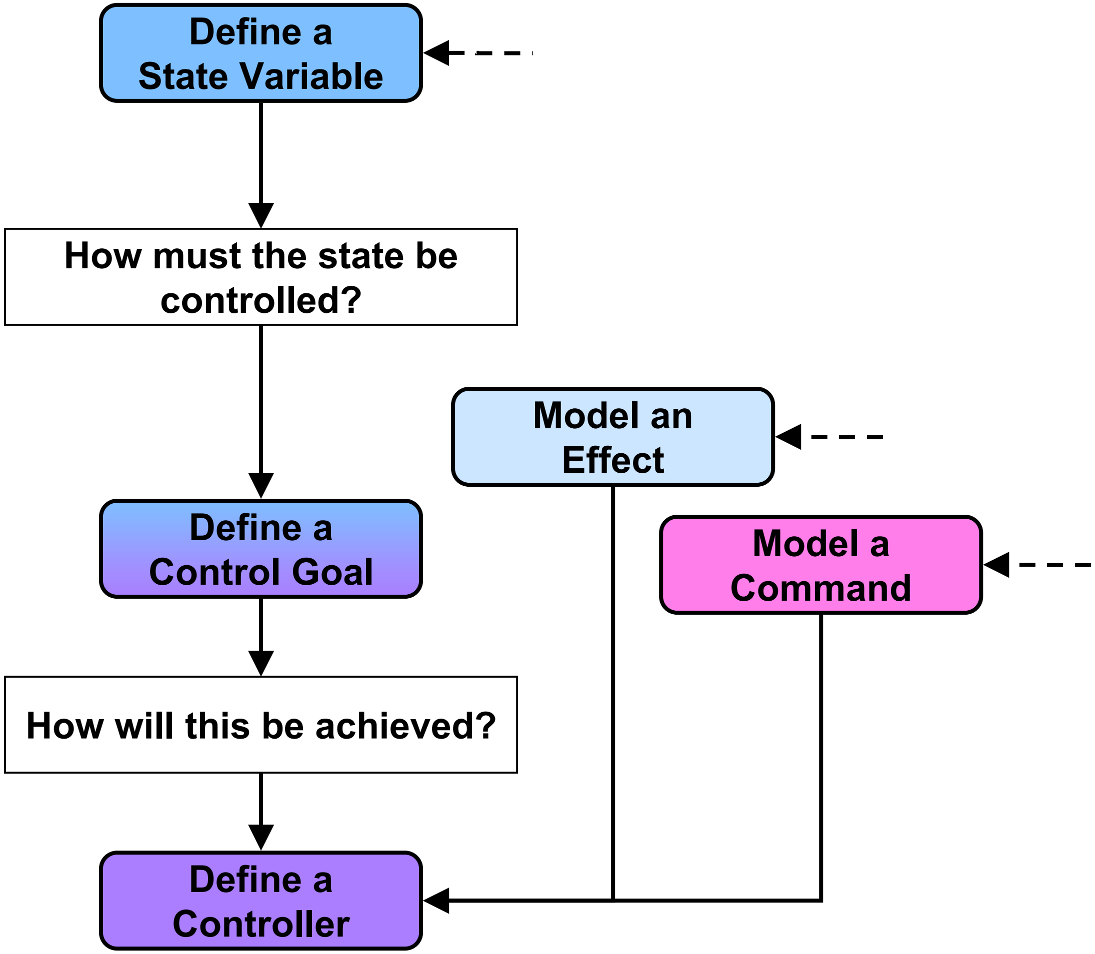


- Estimators are “goal achievers”

State Control

Acting on the System to Control Its State

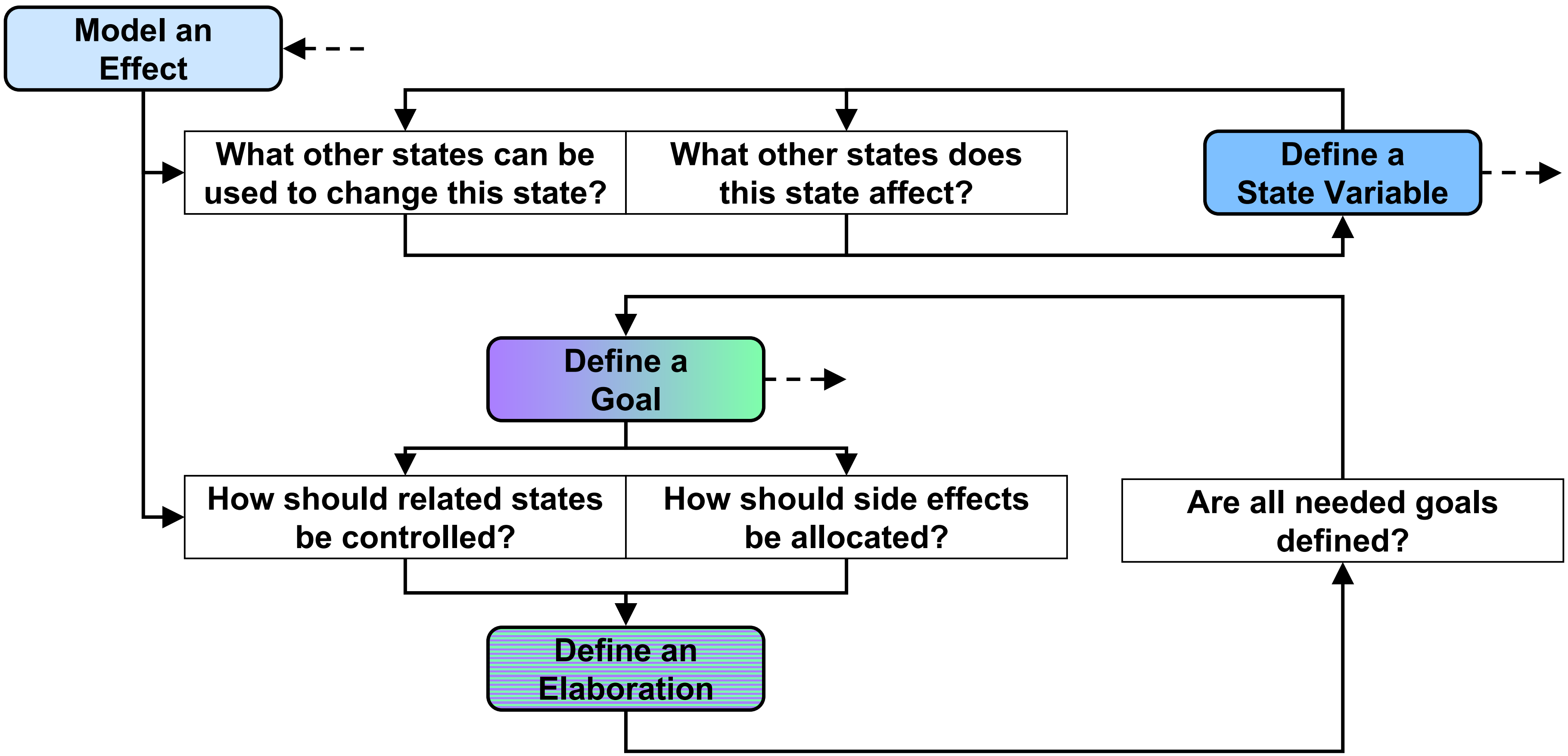
- Models can also suggest how states should be controlled
  - Controllers often use models directly
  - There are usually several ways to control a state



- Controllers are also “goal achievers”

Elaboration

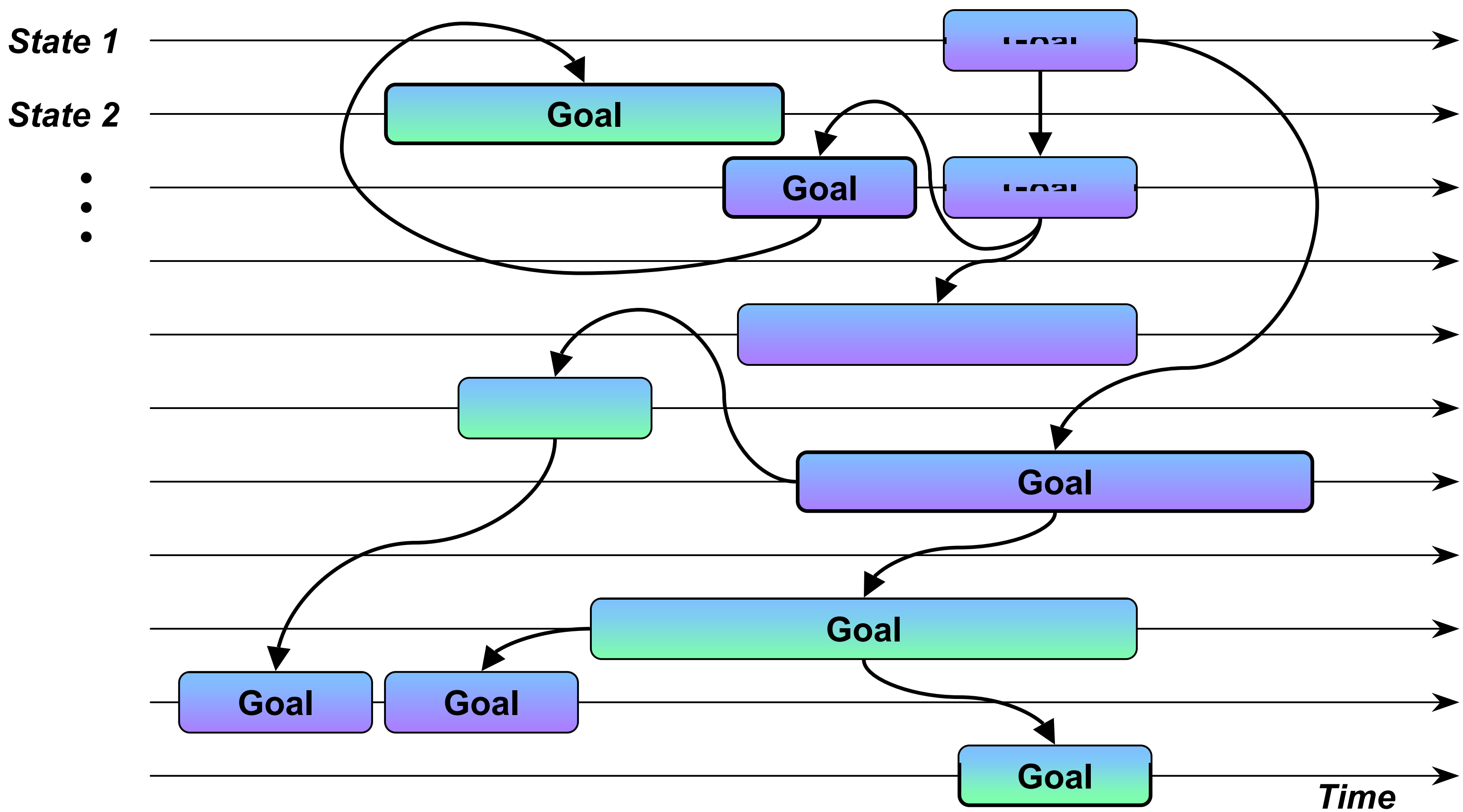
Expressing Intention in Terms of Desired State



- When states can’t be commanded directly, control other states that affect them instead — and don’t forget to allocate side effects
  - Elaboration identifies new goals, and so on

Constraint Networks

- Goals elaborate recursively into constraint networks
- These are scheduled across state timelines, describing a scenario



Following Leads — An Example

Standard Questions:

Common Framework Elements:

What do you want to achieve?

Move rover to rock — ***Goal***

What's the state to be controlled?

Rover position relative to rock — ***State Variable***

What evidence is there for that state?

IMU, wheel rotations, sun sensor, stereo camera — ***Measurements***

What does the stereo camera measure?

Distance to terrain features, light level, camera power (ON/OFF), camera health — ***Measurement Model***

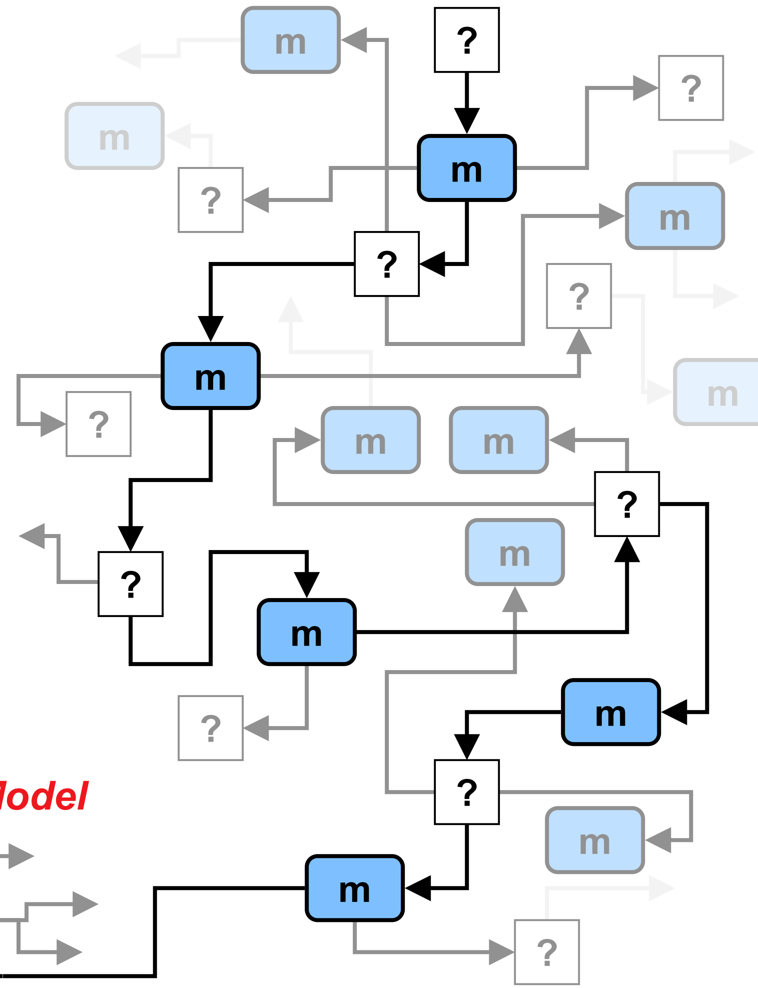
How do you raise the light level?

Wait until the sun is up — ***State Effects Model***

Where is sun relative to horizon?

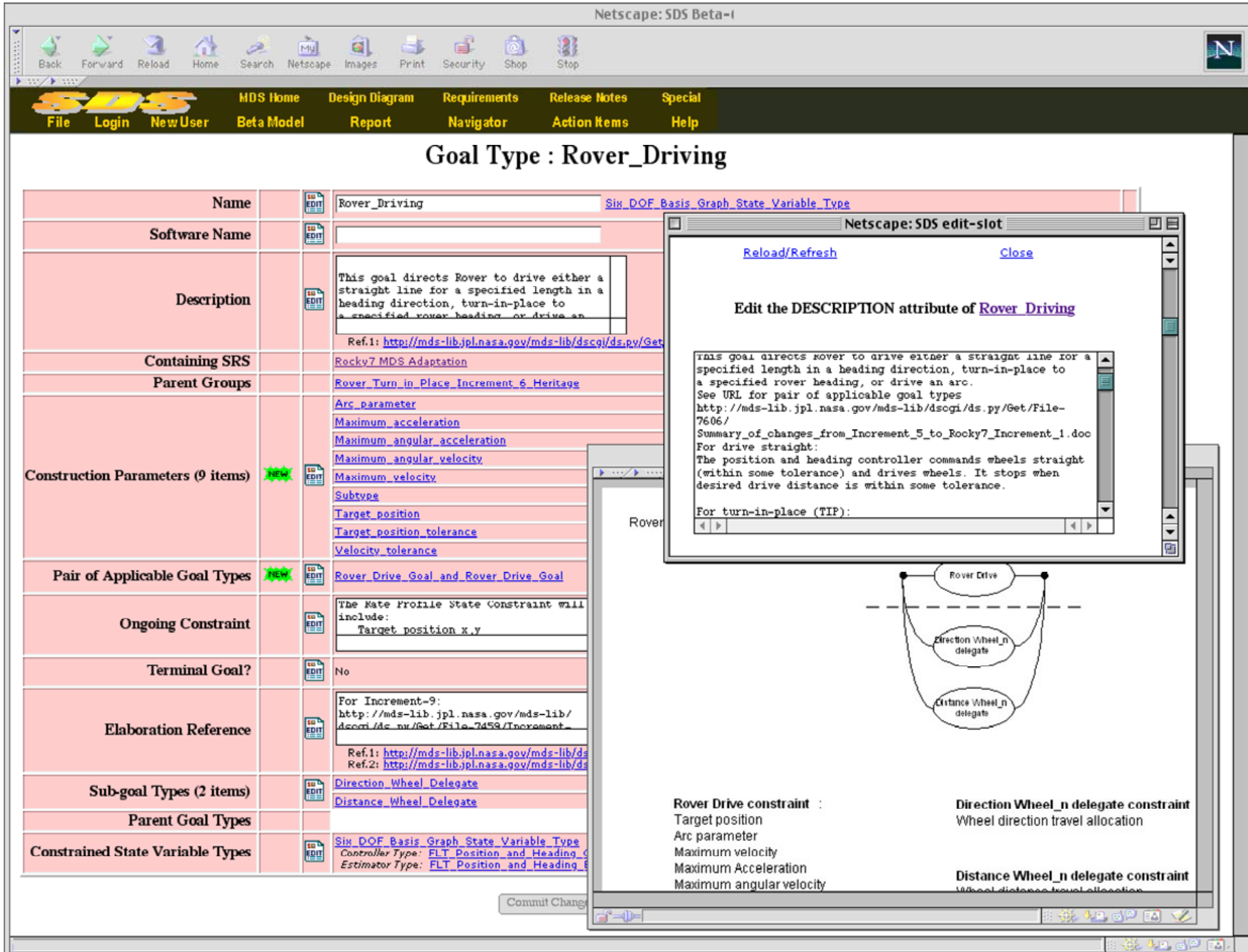
…

Etc.



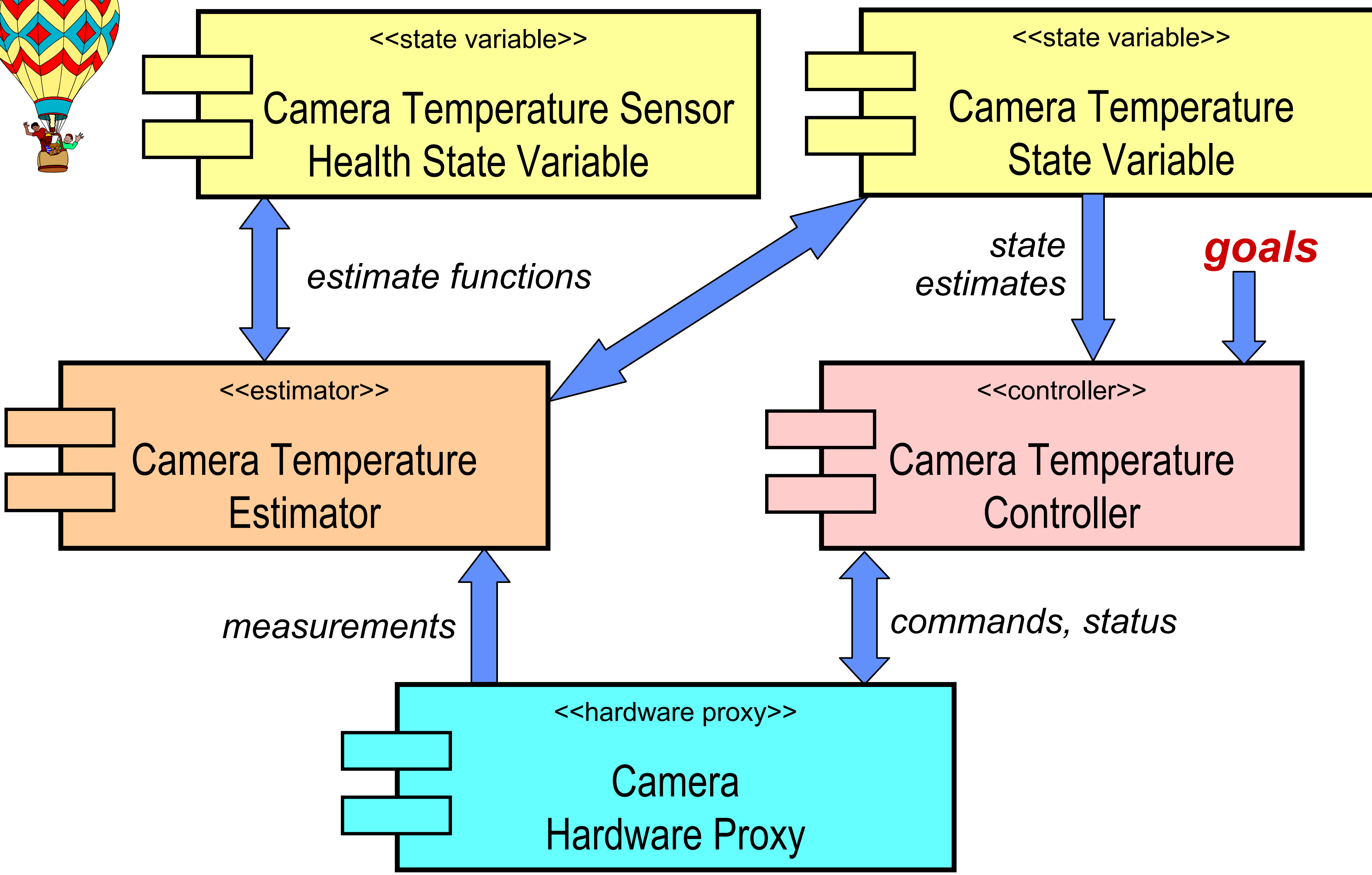
State Database Server
A Tool for State Analysis

Screen display of browser during state analysis



Summary

Inside Thermal, depth 2



Camera Temperature Goal

JPL

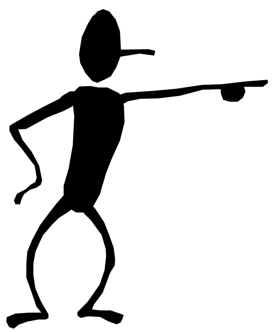
MFRM

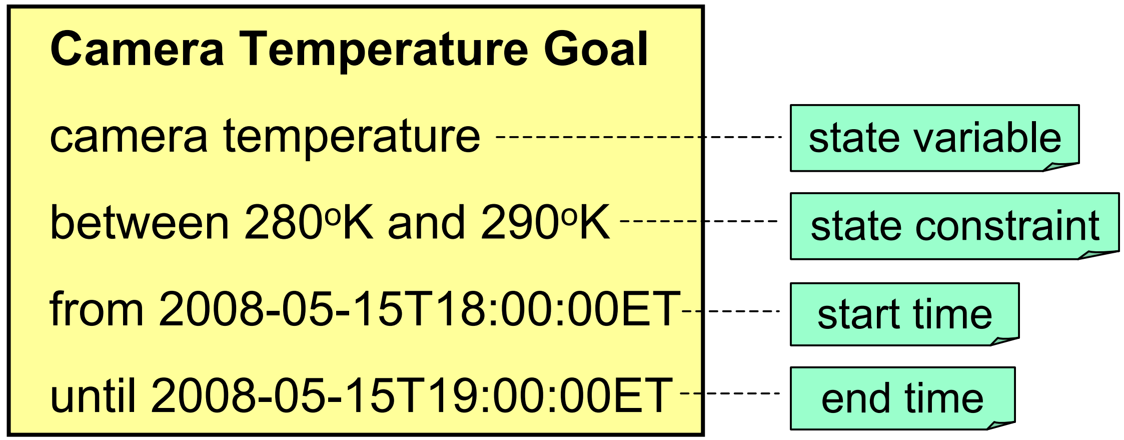
Lander

Thermal
- *depth 1*
- *depth 2*

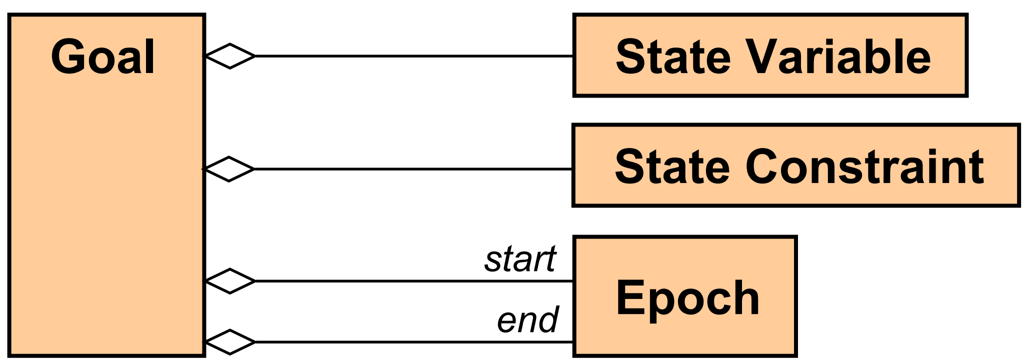
Goal

Adaptation

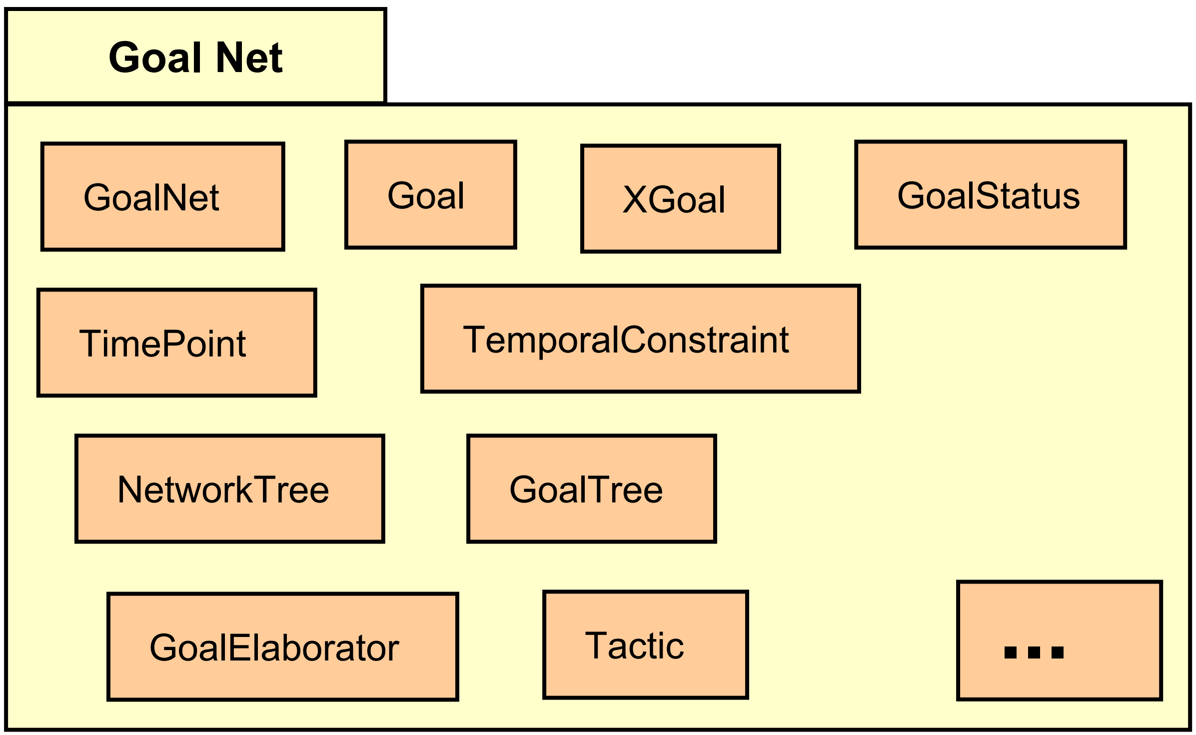




Framework



A Framework Package



Inside Thermal

Camera Temperature Sensor Health State Variable

VH | Cmp | Ep | ...

Camera Temperature State Variable

VH | SI | Cmp | Ep | ...

Camera Temperature Estimator

VH | SI | Cmp | Ep | ...

Camera Temperature Controller

VH | SI | Cmp | Ep | ...

Camera Hardware Proxy

VH | SI | Cmp | Ep | ...

Inside Lander

JPL
MFRM
Lander

Science
SV | Est | Con | HwP | ...

Data Products
SV | Est | Con | HwP | ...

Power
SV | Est | Con | HwP | ...

GNC
SV | Est | Con | HwP | ...

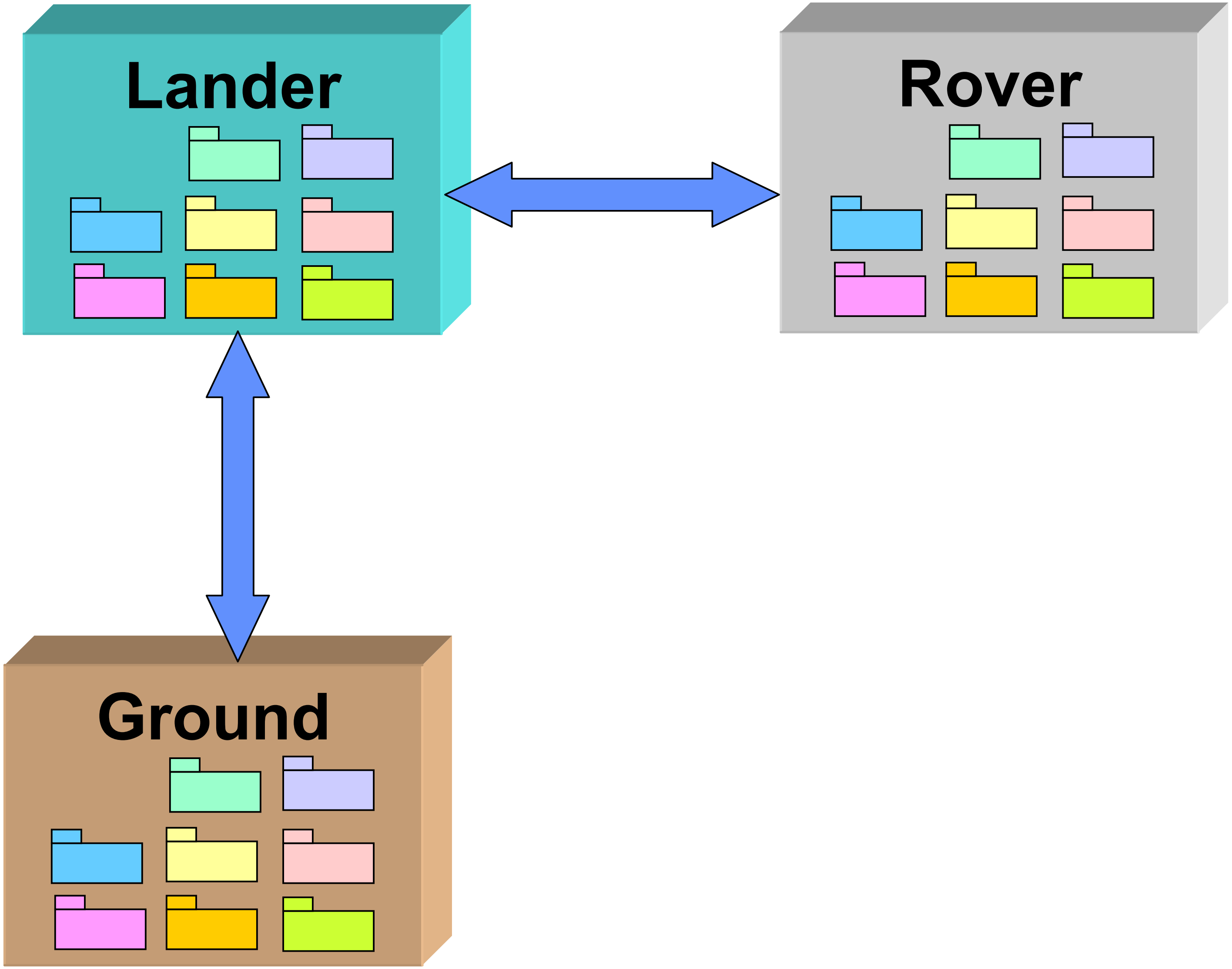
Thermal
SV | Est | Con | HwP | ...

Avionics
SV | Est | Con | HwP | ...

Pyro
SV | Est | Con | HwP | ...

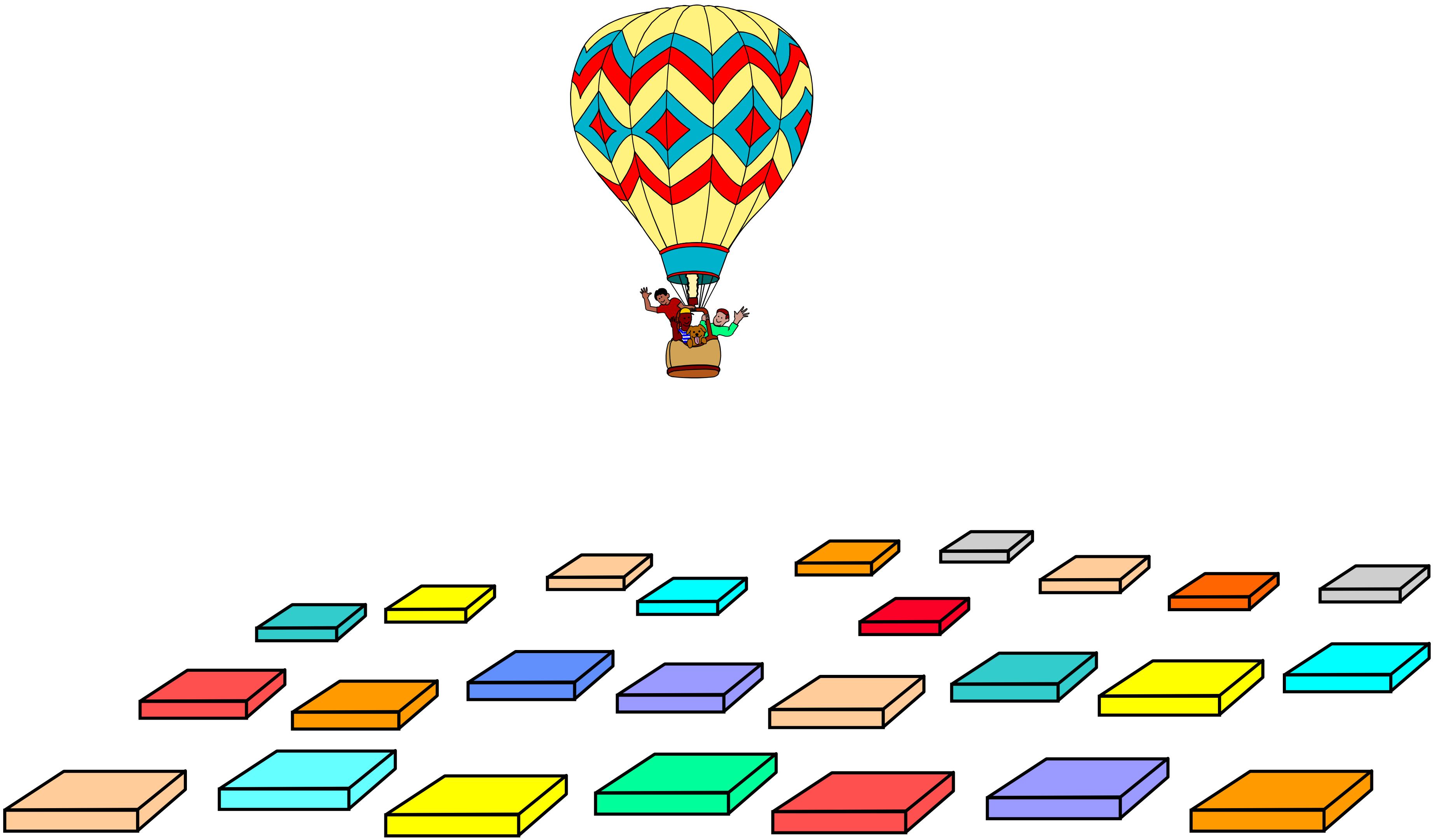
Telecom
SV | Est | Con | HwP | ...

Inside Mission MFRM



JPL
MFRM

25 Missions in Next 10 Years



MDS State/Model Architecture

Estimators interpret measurement and command evidence to estimate state

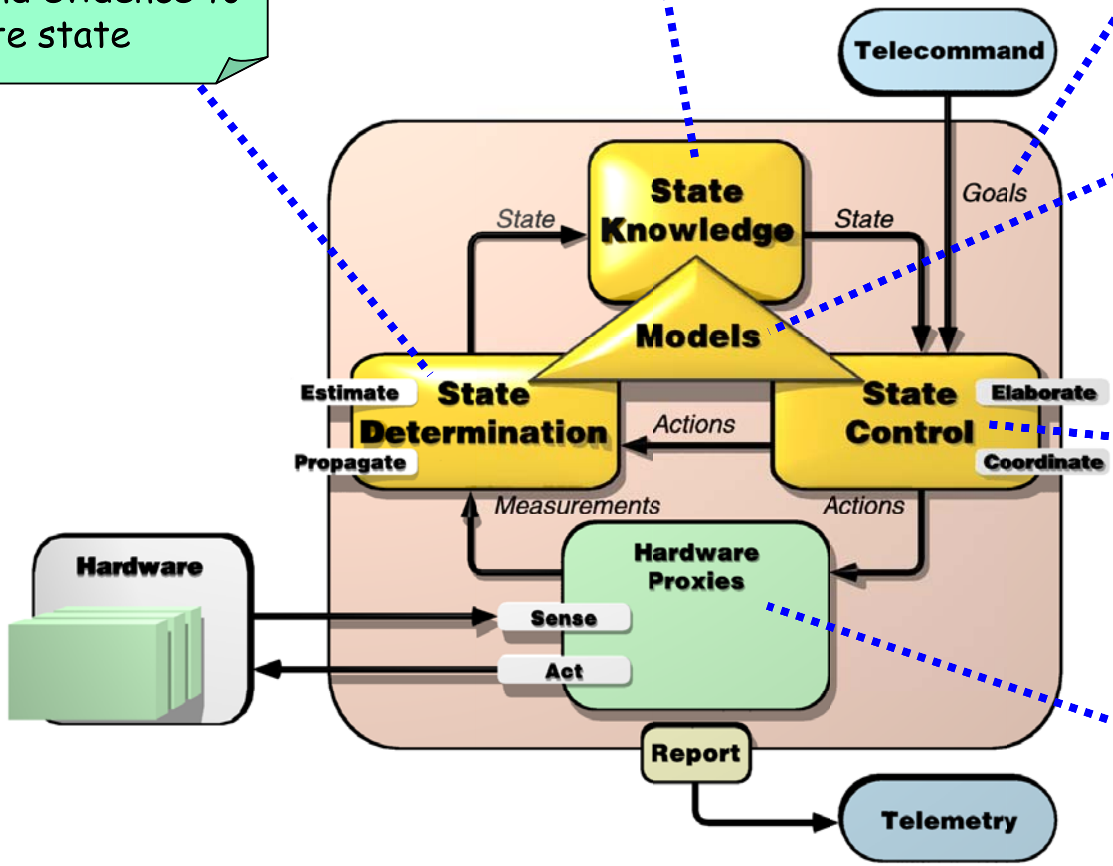
State variables hold state values, including degree of uncertainty

A goal is a constraint on the values of a state variable over a time interval

Models express mission-specific relations among states, commands, and measurements

Controllers issue commands, striving to achieve goals

Hardware proxies provide access to hardware busses, devices, instruments



Advantages of State Analysis

- Captures thorough, unambiguous **requirements**
- Guides a clear **work breakdown**
- Aids collection of **metrics**
- Fosters a **robust design** approach
- Aids cross-checking for **coverage** and **consistency**
- Serves as an **integration** tool at many levels
- Improves **inspectability** and **testability**
- Enables principled **coordination** of the system
- Facilitates increasing **autonomy**
- Assures greater **reusability**

Needs Addressed by MDS

Program Managers
- Continuity across projects
- Investments lead to savings

Project Managers
- Better cost estimates — greater cost control
- More reliable, efficient, and better understood systems

Systems Engineers
- Disciplined and thorough methodology for design
- Easy communication with software engineers — more timely, explicit, complete, and testable requirements

Software Engineers
- Strong formal architecture — "A place for everything, and everything in its place"
- Investment in new design, instead of rehashing old design

Operations
- Reduced overhead for routine operations
- Greater robustness — focus on what to do, not how to do it

Technologists
- More direct path to flight — narrows the TRL gap
- Easier integration with other technologies via standard architecture

Scientists
- Enables more complex missions with autonomy
- Reactive, intelligent data collection and processing

Line managers
- A place to capture institutional experience and knowledge
- Keep a competitive edge

Questions?

“Observations” Slides

- The following slides are to be shown on a second projector, concurrently with parts of the virtual tour
- To synchronize the two screens, match the ‘B’ number with the ‘A’ number

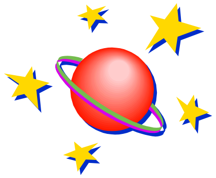
MDS Virtual Tour

- Watch for:
  - architecture
  - frameworks
  - design patterns
  - object-oriented design
  - adaptations
  - MDS terminology (in red)
  - UML‡ terminology (in blue)
- Observations about the sights below will appear here
- Our “altitude” appears here

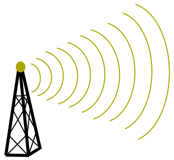
JPL

‡ UML: “a language for visualizing, specifying, constructing and documenting the artifacts of a software intensive system

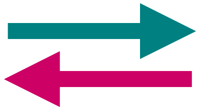
Observations



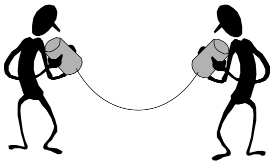
- A "deployment" is:
  - an executable hosted on specific hardware
  - a node in an *interplanetary* network



- Ground ⌧ Lander and Lander ⌧ Rover communication handled uniformly



- "Products" flow in both directions
  - value histories for samples and intervals
  - goals for operation



- Protocols:
  - standard CCSDS protocols, currently
  - internet protocols, future
  - architecture allows any protocol as a plug-in

Observations

- A "discipline" is an area of expertise that builds software assets, sometimes a full subsystem
- Discipline-specific software is organized into "packages" (general-purpose grouping/organizing)
- MDS provides "frameworks" from which subsystems are built
  - shared problems, shared solutions
    - planning, control, telemetry, storage, …
  - subsystems leverage framework software
  - "A framework is an architectural pattern that provides an extensible template for applications within a domain."

Observations

- The system — and its subsystems — is composed of "components"
- A component:
  - is a piece of functionality
  - is the unit of distribution & assembly
  - interacts with other components *only* thru connectors
- Components:
  - enable an *architectural* description
  - eliminate hidden "usage" relations
  - moves coordination/synchronization complexity into connectors

Observations

- Kinds of components:
  - hardware proxies
  - controllers
  - estimators
  - state variables
- Each kind plays a specific role:
  - hardware proxies provide access to hardware
  - controllers strive to achieve goals on state
  - estimators interpret evidence to form estimates of state knowledge
  - state variables hold state knowledge

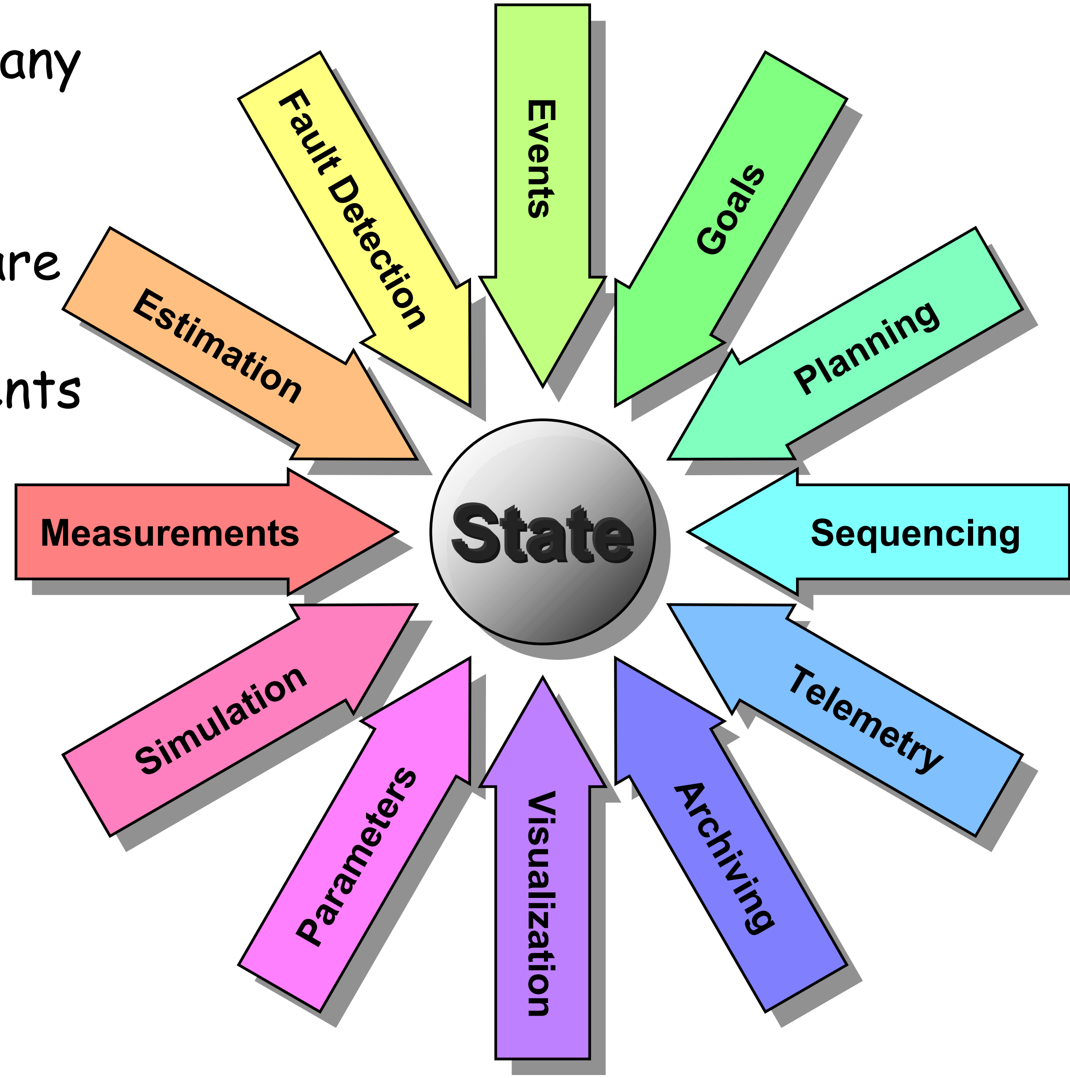
Observation: State is Central

State is a central concept in many mission activities

The function of mission software is to monitor and control a system to meet operators' intents

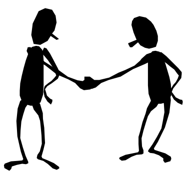
Knowledge of the system is represented over time in **state variables**

Operators' intent, including flight rules and constraints, are expressed as **goals** on system states

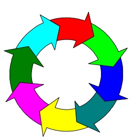


"State Flower", Robert Rasmussen, 1999.

Observations



- Estimation is separated from control
  - distinct roles
  - ensures consistent state seen by all
  - code easier to write, easier to test
  - components are more reusable
- Real-time closed-loop control systems are composed from such components



- Controllers are goal-driven

Observations

- <u>All</u> interactions with a state variable takes place through its interfaces
  - “An interface is a collection of operations that are used to specify a service of a class or component”
- Interface definitions are the end product of a lot of design thought
  - pre-tested for adequacy
  - pre-integrated with other components
- For each interface of a component:
  - some are called from *outside* (e.g. state query)
  - some are called from *inside* (e.g. notification)

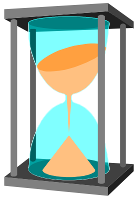
Observations

- An interface defines one or more operations on a component

- "Smart pointers" reduce problems of object ownership and memory leaks

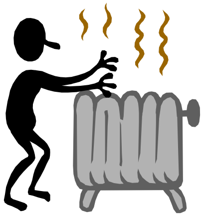
- Scalar quantities, such as temperature, use the "SI units" package
  - not just type-safe, but also unit-safe
  - checked at compile time, not runtime

Observations



- Timelines hold past, present and future
  - Estimate functions describe how state evolved up until 'now'
  - Goals describe how operators *intend* for state to evolve in future
- Timeline is stored in a "value history"
- Data management "policies" control:
  - when to checkpoint
  - what to transport
  - when to compress
  - how much to recover upon restart

Observations

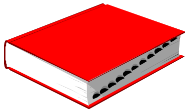


- We're back in Thermal subsystem
- Recall: controllers strive to achieve goals
- Let's look at a goal on camera temperature state

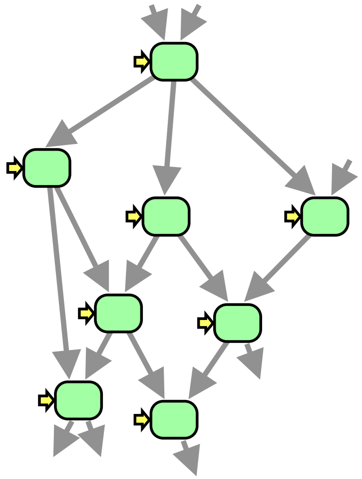
Observations

- A goal is a constraint on the value of a state variable over a time interval

Definition

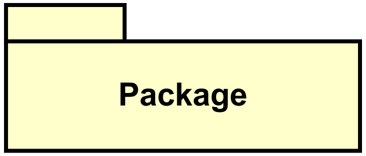


- Goals specify operational intent
  - "what", not "how"
  - goals leave options for selecting actions
  - goals enable *in situ* decision-making
- Goals live in a network that defines parent-child relationships and temporal ordering relationships.

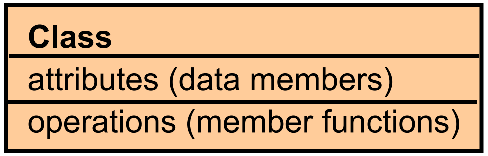


- Adaptations build upon framework software

Observations

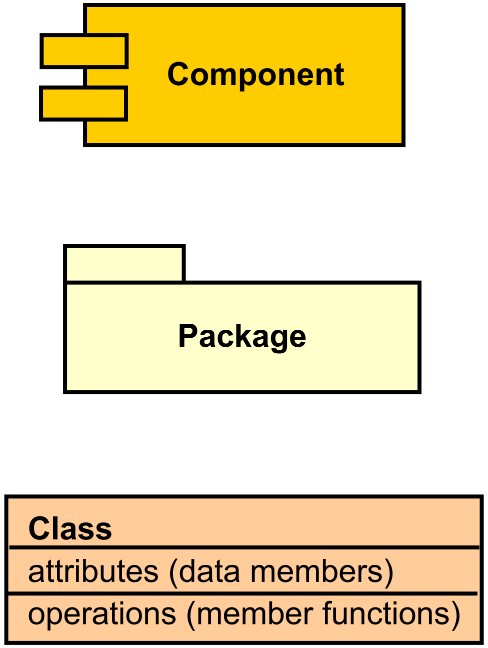


- *A package* organizes multiple classes



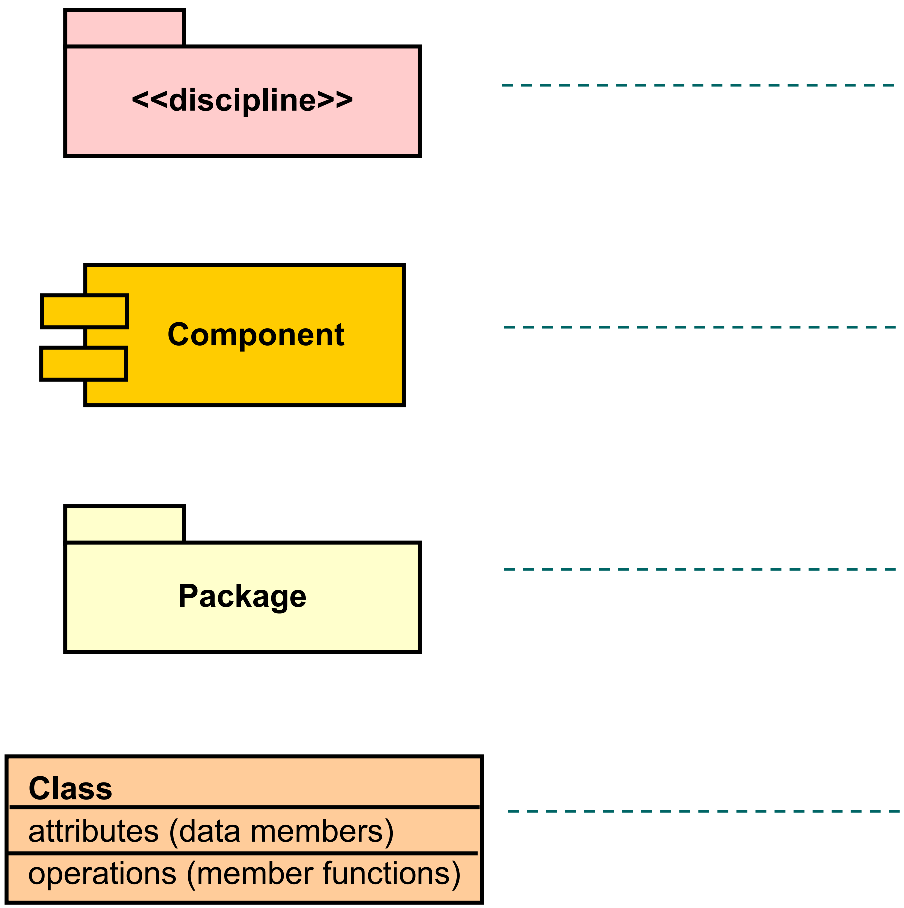
- *A class* defines data and functions

Observations



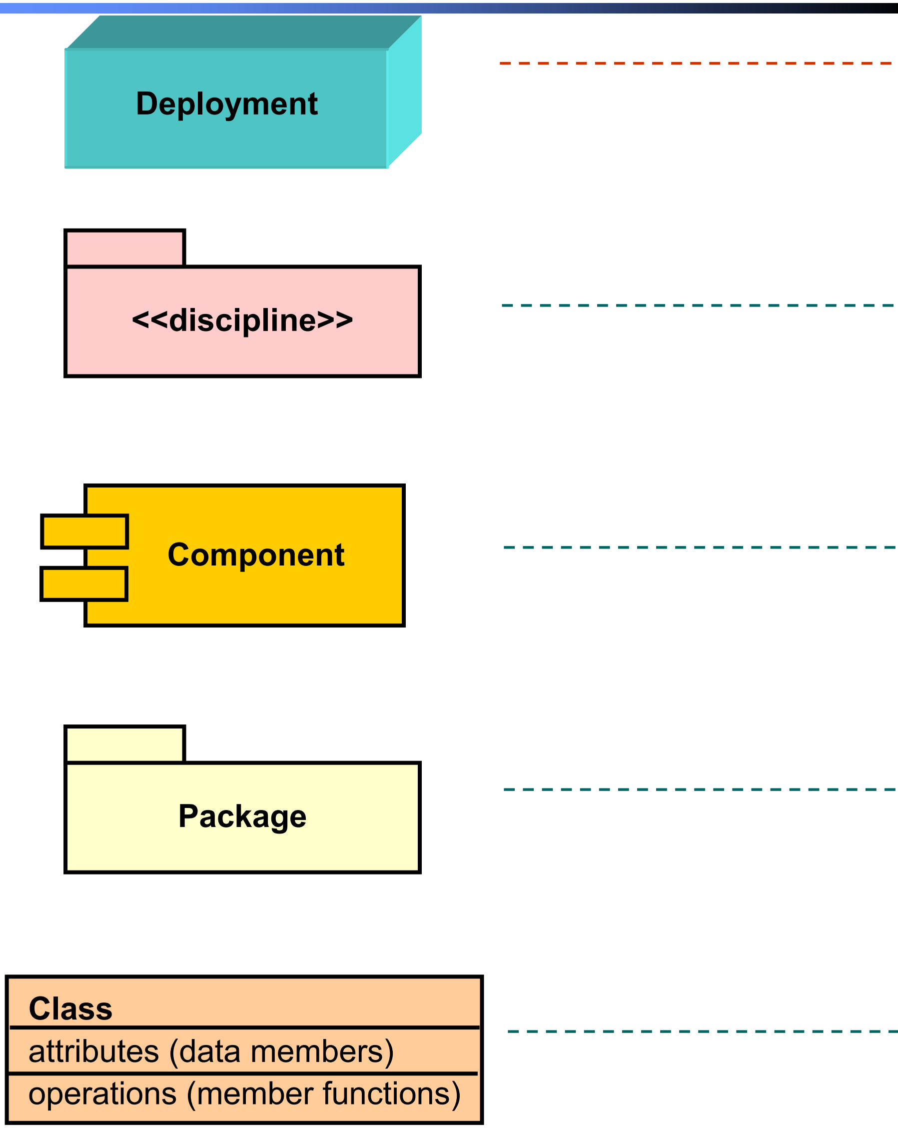
- A *component* uses multiple packages
- A *package* organizes multiple classes
- A *class* defines data and functions

Observations



- A *discipline* builds upon components and other s/w
- A *component* uses multiple packages
- A *package* organizes multiple classes
- A *class* defines data and functions

Observations



- A *deployment* builds upon multiple disciplines
- A *discipline* builds upon components and other s/w
- A *component* uses multiple packages
- A *package* organizes multiple classes
- A *class* defines data and functions

Backup Slides

MDS Framework Packages

Level	Packages
Application Services Level 5	Goal Elaboration Language; State Query; Data Visualization; Simulation; Component Scheduler
State Services Level 4	State Knowledge -state variable -state value; Goal Achiever: -estimator, controller -Measurement, command; Goal Network; Hardware Proxy; Graph State Variable
Complex Services Level 3	Components & Connectors; Value History -sampled history -time-interval history; Data Catalog -collection, entry, event -data product; Data Transport -sender, receiver -session, request
Simple Services Level 2	Embedded web Server & client; Event Log Facility; Naming Services; Time Services; Data Mgmt Policy; CCSDS File Delivery Protocol
Primitive Services Level 1	Initialization & Finalization; Data Serialization; Exception Classes; Standard Utility classes; Sequential Estimation; Graph Library; Physics Library: -SI Units -Coordinate systems -Position, velocity & acceleration; Math Library -Linear algebra -Probability dist. -Polynomials -6-DOF classes
OS Services Level 0	C++ Standard Library; Unit Testing Package; Adaptive Communication Environment; Real Time Operating System